EXHIBIT E
(2 of 2)

EXHIBIT 3

United States Patent [19]

Cox et al.

[11] **Patent Number: 5,008,829**

[45] **Date of Patent: Apr. 16, 1991**

[54] **PERSONAL COMPUTER POWER SUPPLY**

[75] Inventors: **Roger L. Cox**, Plantation; **Michael J. DeLoye**, Boynton Beach; **Robert L. Myers**, Boca Raton, all of Fla.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: 536,751

[22] Filed: Jun. 14, 1990

[51] Int. Cl.[5] G06F 15/20

[52] U.S. Cl. 364/480; 363/84; 363/125

[58] Field of Search 363/84, 125, 98; 364/480

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,873,618 10/1989 Frederick et al. 363/98

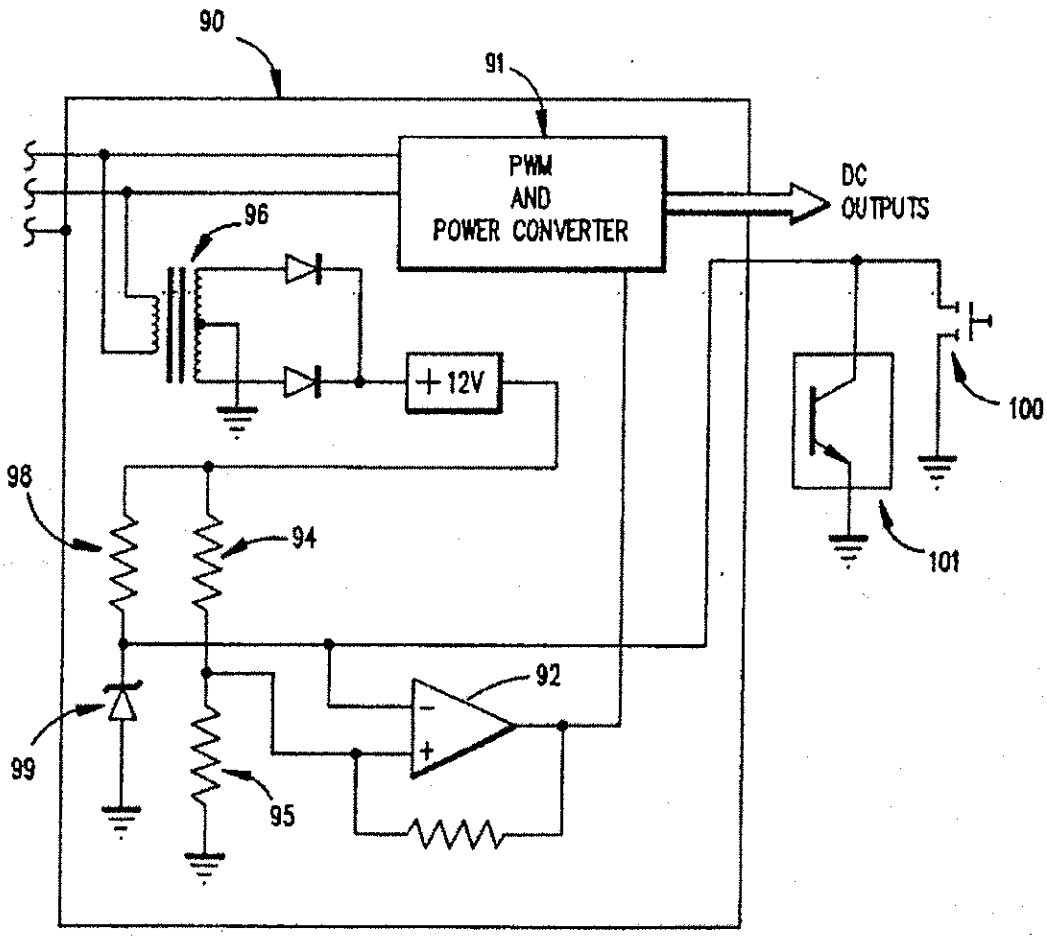
Primary Examiner—Salvatore Cangialosi

Attorney, Agent, or Firm—Daniel E. McConnell

[57] **ABSTRACT**

This invention relates to personal computers, and more particularly to personal computer power supplies for supplying electrical power to electrically operated components which manipulate or store digital data. The power supply has a controllable component for responding to the presence and absence of a low voltage direct current electrical signal by enabling and disabling the supply of electrical power to the data processing and storage components, and a signal generator circuit operatively connected with the controllable component and with an alternating current electrical main supply for controllably deriving from the main supply a low voltage direct current signal for delivery to the controllable component, whereby a user of the microcomputer may control energization of the electrically powered data processing and storage components by controlling the application of the low voltage direct current signal from the signal generator circuit to the controllable component.

10 Claims, 4 Drawing Sheets



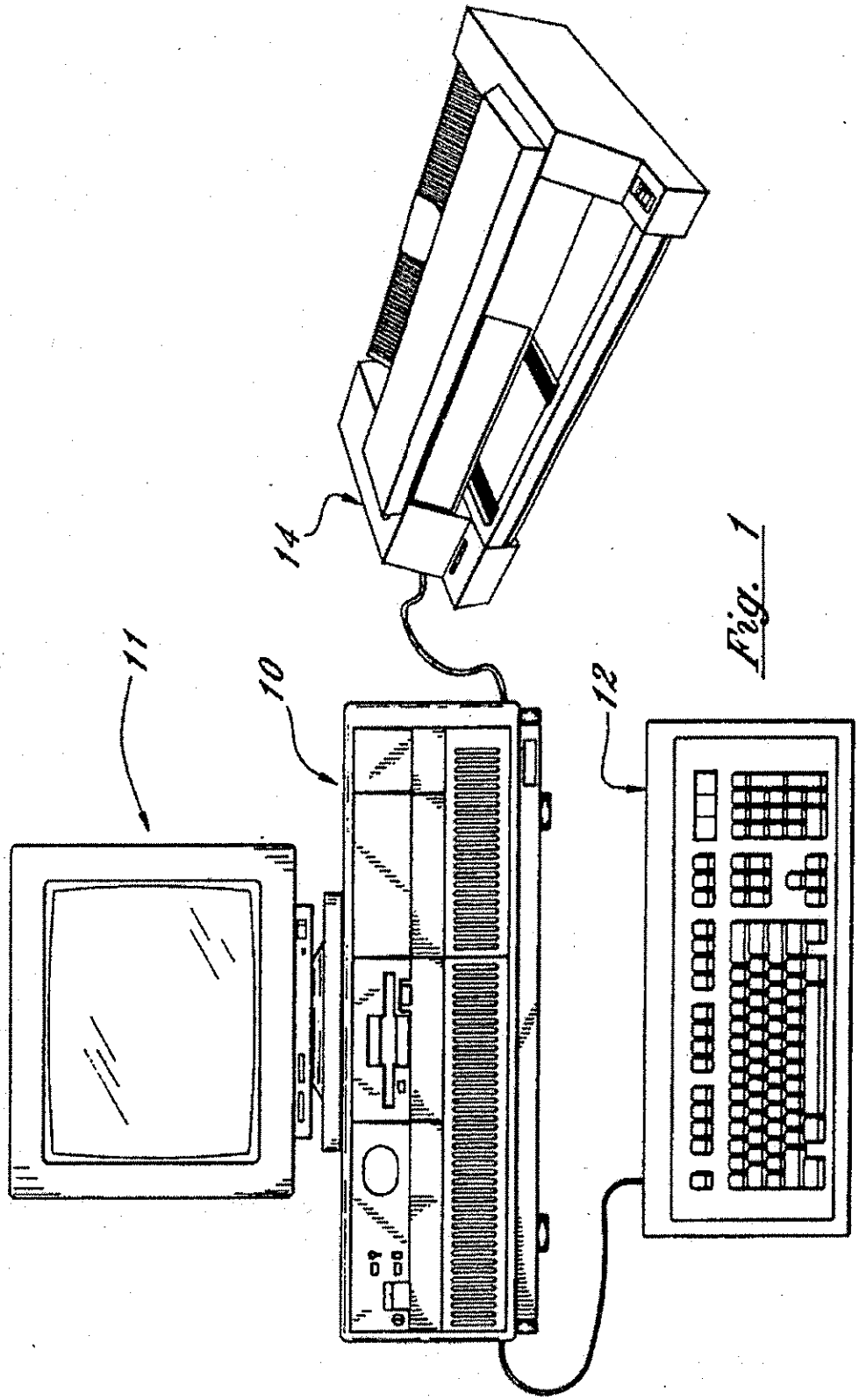
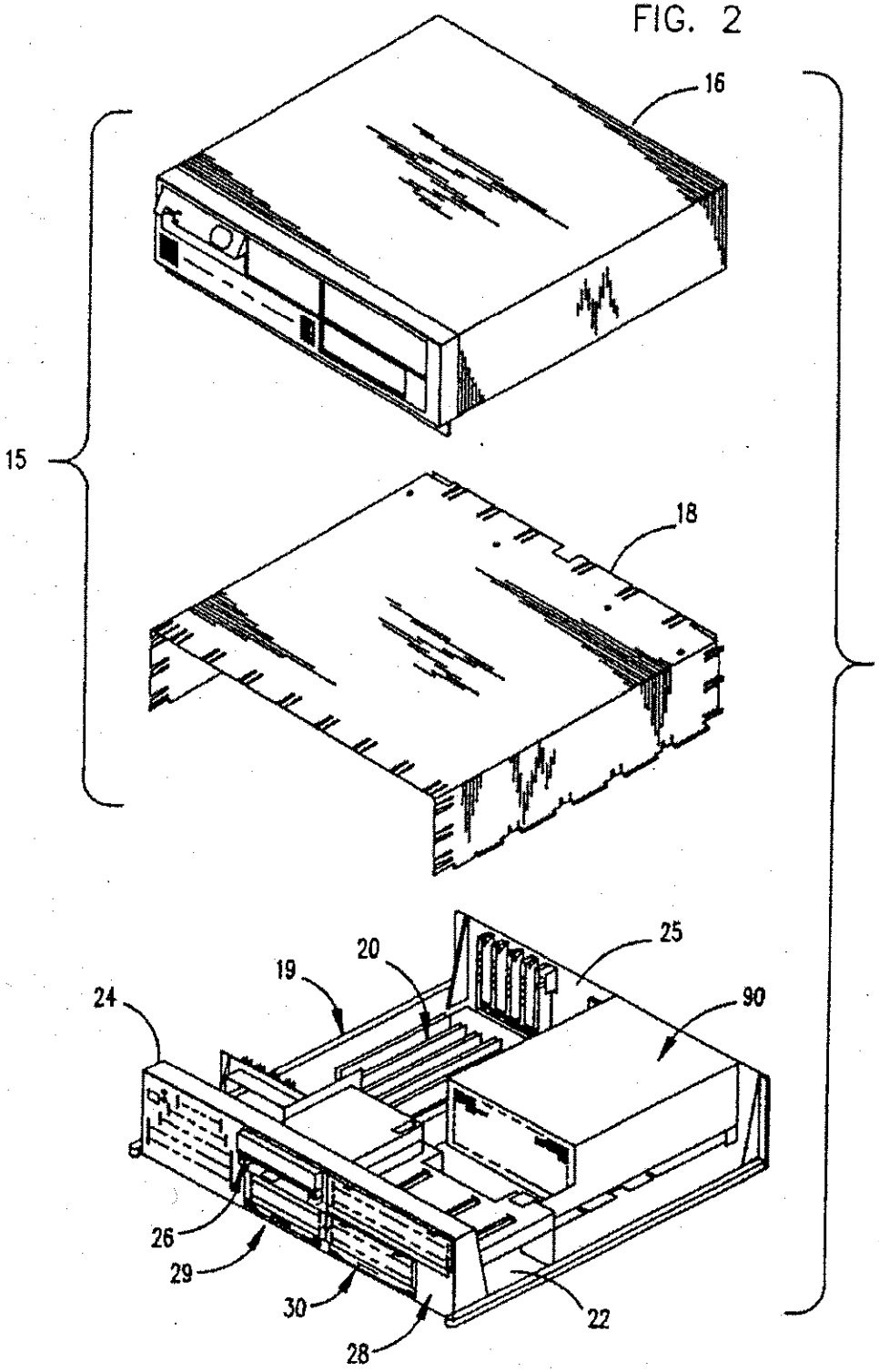


Fig. 1

FIG. 2



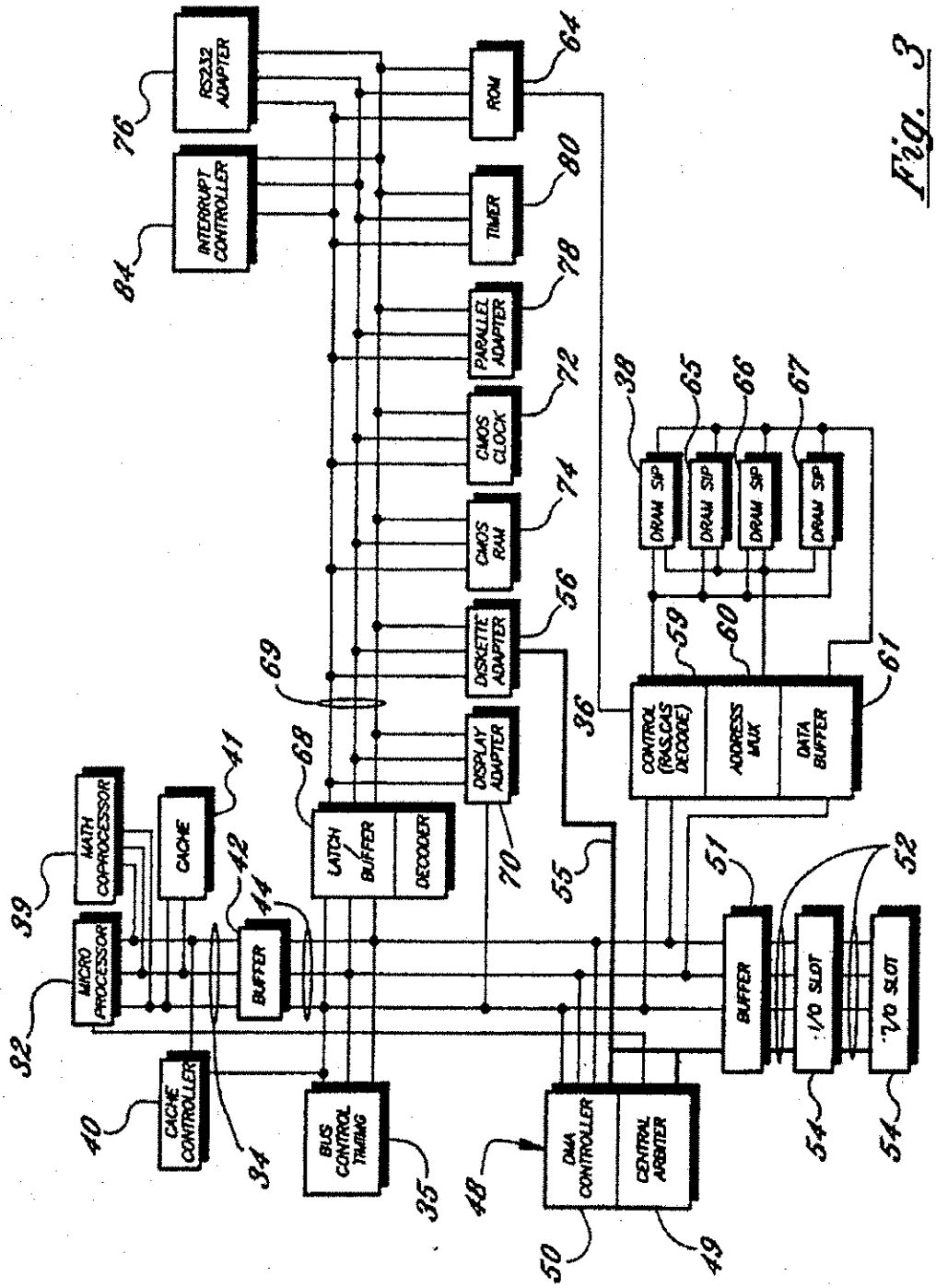
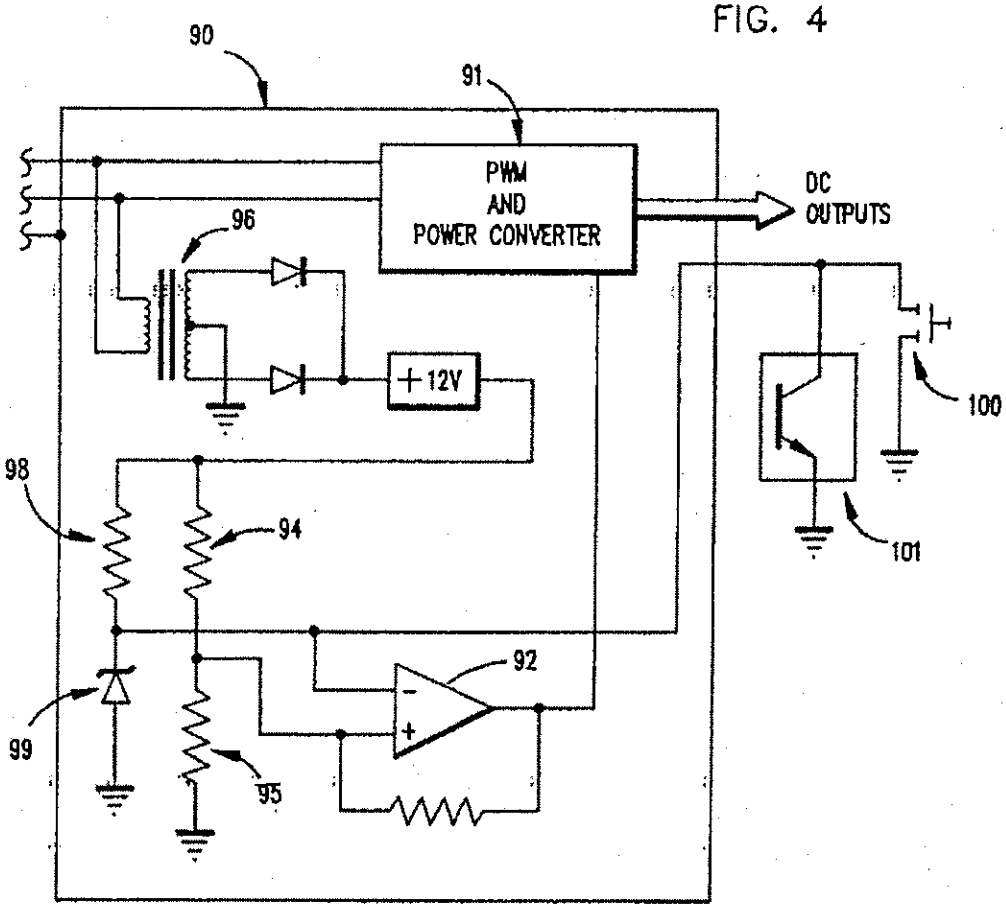

RS232 ADAPTER
INTERRUPT CONTROLLER
ROM
TIMER
PARALLEL ADAPTER
CMOS CLOCK
CMOS RAM
DISKETTE ADAPTER
DISPLAY ADAPTER
DRAM SIP
DRAM SIP
DRAM SIP
DRAM SIP
CONTROL (RAS,CAS DECODE)
ADDRESS MUX
DATA BUFFER
MATH COPROCESSOR
CACHE
LATCH BUFFER
DECODER
MICRO PROCESSOR
BUFFER
BUFFER
I/O SLOT
I/O SLOT
CACHE CONTROLLER
BUS CONTROL TIMING
DMA CONTROLLER
CENTRAL ARBITER
76
84
64
80
78
72
74
56
38
65
66
67
69
68
70
55
96
59
60
61
39
41
42
44
51
52
32
40
34
35
48
50
49
54
54

Fig. 3

FIG. 4


90
91
PWM
AND
POWER CONVERTER
DC
OUTPUTS
96
+12V
100
98
94
101
92
99
95

PERSONAL COMPUTER POWER SUPPLY

FIELD AND BACKGROUND OF INVENTION

This invention relates to personal computers, and more particularly to personal computer power supplies for supplying electrical power to electrically operated components which manipulate or store digital data.

Personal computer systems in general and IBM personal computers in particular have attained widespread use for providing computing capability to many segments of today's modern society. Personal computer systems can usually be defined as a desk top, floor standing, or portable microcomputer that consists of a system unit having a single system processor and associated volatile and non-volatile memory, a display monitor, a keyboard, one or more diskette drives, a fixed disk storage, and an optional printer. One of the distinguishing characteristics of these systems is the use of a motherboard or system planar to electrically connect these components together. These systems are designed primarily to give independent computing capability to a single user and are inexpensively priced for purchase by individuals or small businesses. Examples of such personal computer systems are IBM's PERSONAL COMPUTER AT and IBM's PERSONAL SYSTEM/2 Models 25, 30, 50, 55, 60, 65, 70 and 80.

These systems can be classified into two general families. The first family, usually referred to as Family I Models, use a bus architecture exemplified by the IBM PERSONAL COMPUTER AT and other "IBM compatible" machines. The second family, referred to as Family II Models, use IBM's MICRO CHANNEL bus architecture exemplified by IBM's PERSONAL SYSTEM/2 Models 50 through 80. The Family I models typically have used the popular INTEL 8088 or 8086 microprocessor as the system processor. These processors have the ability to address one megabyte of memory. The Family II models typically use the high speed INTEL 80286, 80386, and 80486 microprocessors which can operate in a real mode to emulate the slower speed INTEL 8086 microprocessor or a protected mode which extends the addressing range from 1 megabyte to 4 Gigabytes for some models. In essence, the real mode feature of the 80286, 80386, and 80486 processors provide hardware compatibility with software written for the 8086 and 8088 microprocessors.

Electrical power for energizing the components of such personal computers is conventionally supplied by power supplies which use control logic, switching transistors, power transformers, rectifiers and filters to convert electrical power from an available line voltage and current, such as the 110 volt 60 hertz current supplied in the United States, to the direct voltages and currents required for operation of the personal computer. It has been conventional to control operation of such power supplies by switching the supply voltage to the power supply. In the example given, such switching is accomplished by turning on and off the 110 volt 60 hertz mains supply current. As will be appreciated, such switching of the generally supplied service voltage requires switches which are at least somewhat substantial as compared to the significantly lower voltages and current typically used with the operating components of the personal computer.

It has been proposed heretofore, as in Summerlin U.S. Pat. No. 4,723,269, to use a lower level voltage to control the operation of a personal computer power supply. In the Summerlin disclosure, to which the interested reader is referred, a telephone ring detector acts through optically coupled semiconductor devices to control passage of the relatively high voltage supply current. However, such circuitry continues to require manual control over computer power to be exercised by manipulation of a relatively high voltage switch.

BRIEF DESCRIPTION OF INVENTION

With the foregoing discussion particularly in mind, it is an object of this invention to protect a user of a personal computer against unnecessary exposure to the relatively high voltages of sources of supply to the computer. In realizing this object of the present invention, provision is made for the delivery of a relatively low voltage signal which may be manually or remotely controlled to control the operation of a pulse width modulator control component in a power supply without any necessity of providing a high voltage optically isolated or electromechanical switch to directly control the mains supply voltage.

Yet a further object of this invention is to enable control over the operation of a personal computer with logic level voltage, low current, electrical signals. In realizing this object of the present invention, the possibility is opened of establishing remote control over the supply of electrical power to the operating components of a computer in a manner consistent with the manual control used by an operator.

BRIEF DESCRIPTION OF DRAWINGS

Some of the objects of the invention having been stated, other objects will appear as the description proceeds, when taken in connection with the accompanying drawings, in which:

FIG. 1 is a perspective view of a personal computer embodying this invention;

FIG. 2 is an exploded perspective view of certain elements of the personal computer of FIG. 1 including a chassis, a cover, an electromechanical direct access storage device and a planar board and illustrating certain relationships among those elements;

FIG. 3 is a schematic view of certain components of the personal computer of FIGS. 1 and 2; and

FIG. 4 is a circuit diagram of certain power supply and control elements used in the computer of FIGS. 1 through 3.

DETAILED DESCRIPTION OF INVENTION

While the present invention will be described more fully hereinafter with reference to the accompanying drawings, in which a preferred embodiment of the present invention is shown, it is to be understood at the outset of the description which follows that persons of skill in the appropriate arts may modify the invention here described while still achieving the favorable results of this invention. Accordingly, the description which follows is to be understood as being a broad, teaching disclosure directed to persons of skill in the appropriate arts, and not as limiting upon the present invention.

Referring now more particularly to the accompanying drawings, a microcomputer embodying the present invention is there shown and generally indicated at 10 (FIG. 1). As mentioned hereinabove, the computer 10 may have an associated monitor 11, keyboard 12 and printer or plotter 14. The computer 10 has a cover 15 formed by a decorative outer member 16 (FIG. 2) and

an inner shield member 18 which cooperate with a chassis 19 in defining an enclosed, shielded volume for receiving electrically powered data processing and storage components for processing and storing digital data. At least certain of these components are mounted on a planar 20 which is mounted on the chassis 19 and provides a means for electrically interconnecting the components of the computer 10 including those identified above and such other associated elements as floppy disk drives, various forms of direct access storage devices, accessory cards or boards, and the like.

The chassis 19 has a base indicated at 22, a front panel indicated at 24, and a rear panel indicated at 25 (FIG. 2). The front panel 24 defines at least one open bay (and in the form illustrated, four bays) for receiving a data storage device such as a disk drive for magnetic or optical disks, a tape backup drive, or the like. In the illustrated form, a pair of upper bays 26, 28 and a pair of lower bays 29, 30 are provided. The upper bays 26, 28 are adapted to receive peripheral drives of a first size (such as those known as 5.25 inch drives) while the lower bays are adapted to receive devices of another size (such as those known as 3.5 inch drives).

Prior to relating the above structure to the present invention, a summary of the operation in general of the personal computer system 10 may merit review. Referring to FIG. 3, there is shown a block diagram of a personal computer system illustrating the various components of the computer system such as the system 10 in accordance with the present invention, including components mounted on the planar 20 and the connection of the planar to the I/O slots and other hardware of the personal computer system. Connected to the planar is the system processor 32 comprised of a microprocessor which is connected by a high speed CPU local bus 34 through a bus control timing unit 35 to a memory control unit 36 which is further connected to a volatile random access memory (RAM) 38. While any appropriate microprocessor can be used, one suitable microprocessor is the 80386 which is sold by INTEL.

While the present invention is described hereinafter with particular reference to the system block diagram of FIG. 3, it is to be understood at the outset of the description which follows that it is contemplated that the apparatus and methods in accordance with the present invention may be used with other hardware configurations of the planar board. For example, the system processor could be an Intel 80286 or 80486 microprocessor and the system bus could be a Microchannel or AT type.

Returning now to FIG. 3, the CPU local bus 34 (comprising data, address and control components) provides for the connection of the microprocessor 32, a math coprocessor 39, a cache controller 40, and a cache memory 41. Also coupled on the CPU local bus 24 is a buffer 42. The buffer 42 is itself connected to a slower speed (compared to the CPU local bus) system bus 44, also comprising address, data and control components. The system bus 44 extends between the buffer 42 and a further buffer 51. The system bus 44 is further connected to a bus control and timing unit 35 and a DMA unit 48. The DMA unit 48 is comprised of a central arbitration unit 49 and DMA controller 50. The buffer 51 provides an interface between the system bus 44 and an optional feature bus such as the MICRO CHANNEL bus 52. Connected to the bus 52 are a plurality of I/O slots 54 for receiving MICRO CHANNEL adapter cards which may be further connected to an I/O device or memory.

An arbitration control bus 55 couples the DMA controller 50 and central arbitration unit 49 to the I/O slots 54 and a diskette adapter 56. Also connected to the system bus 44 is a memory control unit 36 which is comprised of a memory controller 59, an address multiplexor 60, and a data buffer 61. The memory control unit 36 is further connected to a random access memory as represented by the RAM module 38. The memory controller 36 includes the logic for mapping addresses to and from the microprocessor 32 to particular areas of RAM 38. This logic is used to reclaim RAM previously occupied by BIOS. Further generated by memory controller 36 is a ROM select signal (ROMSEL), that is used to enable or disable ROM 64.

While the microcomputer system 10 is shown with a basic 1 megabyte RAM module, it is understood that additional memory can be interconnected as represented in FIG. 3 by the optional memory modules 65 through 67. For purposes of illustration only, the present invention is described with reference to the basic one megabyte memory module 38.

A further buffer 68 is coupled between the system bus 44 and a planar I/O bus 69. The planar I/O bus 69 includes address, data, and control components respectively. Coupled along the planar bus 69 are a variety of I/O adapters and other components such as the display adapter 70 (which is used to drive the monitor 11), a clock 72, nonvolatile RAM 74 herein after referred to as NVRAM, a RS232 adapter 76, a parallel adapter 78, a plurality of timers 80, a diskette adapter 56, an interrupt controller 84, and a read only memory 64. The read only memory 64 includes the BIOS that is used to interface between the I/O devices and the operating system of the microprocessor 32. BIOS stored in ROM 64 can be copied into RAM 38 to decrease the execution time of BIOS. ROM 64 is further responsive (via ROMSEL signal) to memory controller 36. If ROM 64 is enabled by memory controller 36, BIOS is executed out of ROM. If ROM 64 is disabled by memory controller 36, ROM is not responsive to address enquiries from the microprocessor 32 (i.e. BIOS is executed out of RAM).

The clock 72 is used for time of day calculations and the NVRAM is used to store system configuration data. That is, the NVRAM will contain values which describe the present configuration of the system. For example, NVRAM contains information describing the capacity of a fixed disk or diskette, the type of display, the amount of memory, time, date, etc. Of particular importance NVRAM will contain data (can be one bit) which is used by memory controller 36 to determine whether BIOS is run out of ROM or RAM and whether to reclaim RAM intended to be used by BIOS RAM. Furthermore, these data are stored in NVRAM whenever a special configuration program, such as SET Configuration, is executed. The purpose of the SET Configuration program is to store values characterizing the configuration of the system to NVRAM.

There is also mounted within the shielded enclosure and associated with the above described components of the computer a power supply (indicated at 90 in FIG. 2) for supplying electrical power to the components of the computer 10. Preferably, and as is generally known and applied in the field of personal computers, the power supply 90 is a pulse width modulation switching power supply for connection with an alternating current electrical main supply and for supplying direct current elec-

trical power to the data processing and storage components for enabling operation thereof. Such, power supplies have been described, for example, in "The Winn Rosch Hardware Bible", published by Brady of New York under a copyright date of 1989. As there described, power supplies of the switching type are more efficient and often less expensive than their linear kin. While designs vary, the typical switching supply first converts incoming 60 Hz utility power to a much higher frequency of pulses, in the range of 20 kHz. At the same time that the electrical current is increased in frequency, it is regulated by a digital technique known as pulse width modulation. The duration of each power pulse is varied in response to the needs of the computer circuitry being supplied. The width of the pulses is controlled by electronically switching the current flow on and off, hence the name of the technique. Such switching typically occurs in a special purpose chip designed for such a function and operating with a number of other circuit elements not here shown or described. Such description is omitted here because it is believed well known to persons of ordinary skill in the relevant arts. The pulses are reduced in voltage by a transformer and turned into pure direct current by rectification and filtering.

Switching power supplies earn their efficiency and lower cost in two ways: Switching regulation is more efficient because less power is turned into heat. Instead of dissipating energy, the switching regulator switches all current flow on and off. In addition, high frequencies require smaller, less expensive transformers and filtering circuits. Nearly all of today's personal computers use switching power supplies.

In conventional switching power supplies as described immediately above, control over whether any voltage is delivered to the operating components of the computer is exercised by turning the normal utility voltage supply on and off. In the United States, such utility mains typically supply 110 volt 60 Hz alternating electrical current. The dangers to a potential user, and the safety regulations imposed by such bodies as Underwriters Laboratories, are well known to designers of personal computers.

In accordance with this invention, the power supply comprises a controllable component for responding to the presence and absence of a low voltage direct current electrical signal by enabling and disabling the supply of electrical power to the data processing and storage components by controlling the "on" or "off" state of the pulse width modulator, and a signal generator circuit operatively connected with the controllable component and with an alternating current electrical main supply for controllably deriving from the main supply a low voltage direct current signal for delivery to said controllable component. By reason of this interconnection, a user of the computer may control energization of the electrically powered data processing and storage components by controlling the application of a low voltage direct current signal from the signal generator circuit to the controllable component. In the illustrated embodiment, the controllable component is a pulse width modulator control circuit (described generally hereinabove and indicated at 91 in FIG. 4) which may be of conventional form such as is available from Motorola as its SG1525A/1527A series of devices. The signal generator circuit comprises several elements, including a comparator 92 and first and second signal circuits for supplying signals to the comparator.

The comparator 92 receives from a first signal circuit a low voltage direct current reference signal established by a voltage divider 94, 95 supplied with rectified current transformed from the main supply voltage by a suitable small transformer 96. The comparator 92 also receives a controlled low voltage enable/disable signal varying between a first voltage and a second, higher voltage. The enable/disable signal is established by a regulated voltage divider formed by a resistor 98 and an associated zener diode 99, to which is connected a current drain in the form of a switch device for grounding out the point of connection between the resistor and diode. The switch device, which functions at a low, logic level voltage (typically 5 volts) as distinguished from the voltage of the main supply (typically on the order of 100 volts or more), may be in the form of a manually operable switch 100 or a computer logic operable switch 101. In either instance, operation of the switch 100 or 101 functions for selecting between the first and second voltages for the enable/disable signal. Responding to the change in voltage of signals applied, the comparator 92 either delivers a signal forward to a shutdown or inhibit pin of the control circuit 91 or does not deliver a signal and thereby controls the power delivered to the electrically powered data processing and storage components of the computer. The computer logic operable switch 101 may, for example, be associated with a telecommunications device for enabling remote control over the power on and power off states of the computer 10.

It is to be noted that the control occurs on the secondary, or output, side of the primary power transformer of the power supply, where the pulse width modulator component is connected, so that a user is isolated from dangerous voltages.

As will be appreciated, a user of the computer 10 may control the power on power off functions by manipulating the manual switch 100, and thereby use the computer while avoiding exposure to the higher voltages and currents typically supplied through the main electrical supply obtained from a power distribution company or utility. In the drawings and specifications there has been set forth a preferred embodiment of the invention and, although specific terms are used, the description thus given uses terminology in a generic and descriptive sense only and not for purposes of limitation.

What is claimed is:

1. A microcomputer comprising:

electrically powered data processing and storage components for processing and storing digital data, and

a pulse width modulation switching power supply for connection with an alternating current electrical main supply and for supplying direct current electrical power to said data processing and storage components for enabling operation thereof, said power supply comprising:

a controllable component for responding to the presence and absence of a low voltage direct current electrical signal by enabling and disabling the supply of electrical power to said data processing and storage components, and

a signal generator circuit operatively connected with said controllable component and with an alternating current electrical main supply for controllably deriving from the main supply a low voltage direct current signal for delivery to said controllable component, whereby a user of the microcomputer

may control energization of the electrically powered data processing and storage components by controlling the application of said low voltage direct current signal from said signal generator circuit to said controllable component.

2. A microcomputer according to claim 1 wherein said controllable component comprises a pulse width modulator control circuit.

3. A microcomputer according to claim 1 wherein said signal generator circuit comprises a comparator, a first signal circuit for supplying to the comparator a low voltage direct current reference signal, and a controllable second signal circuit for controllably supplying to the comparator a low voltage enable/disable signal varying between a first voltage and a second, higher voltage, said comparator when supplied with said reference signal and said enable/disable signal having said first voltage delivering to said controllable component said first mentioned signal.

4. A microcomputer according to claim 3 wherein said first signal circuit comprises a voltage divider for establishing a preselected voltage level for said reference signal.

5. A microcomputer according to claim 3 wherein said second signal circuit comprises a regulated voltage divider for establishing a preselected, regulated voltage level and current drain for said enable/disable signal.

6. A microcomputer according to claim 5 wherein said second signal circuit comprises a manually operable switch for selecting between the first and second voltages for the enable/disable signal and thereby for controlling the power delivered to said electrically powered data processing and storage components.

7. A microcomputer according to claim 5 wherein said second signal circuit comprises a computer logic operable switch for selecting between the first and second voltages for the enable/disable signal and thereby for controlling the power delivered to said electrically powered data processing and storage components.

8. A microcomputer comprising:

electrically powered data processing and storage components for processing and storing digital data, and

a pulse width modulation switching power supply for connection with an alternating current electrical main supply and for supplying direct current electrical power to said data processing and storage components for enabling operation thereof, said power supply comprising:

a controllable pulse width modulator for responding to the presence and absence of a low voltage direct current electrical signal by enabling and disabling the supply of electrical power to said data processing and storage components, and

a signal generator circuit operatively connected with said controllable component and with an alternating current electrical main supply for controllably deriving from the main supply a low voltage direct current signal for delivery to said controllable component, said signal generator circuit comprising a comparator, a first signal circuit for supplying to the comparator a low voltage direct current reference signal, and a controllable second signal circuit for controllably supplying to the comparator a low voltage enable/disable signal varying between a first voltage and a second, higher voltage, said comparator when supplied with said reference signal and said enable/disable signal having said first voltage delivering to said controllable component said first mentioned signal, whereby a user of the microcomputer may control energization of the electrically powered data processing and storage components by controlling the application of said low voltage direct current signal from said signal generator circuit to said controllable component.

9. A microcomputer according to claim 8 wherein said second signal circuit comprises a regulated voltage divider for establishing a preselected, regulated voltage level and current drain for said enable/disable signal, and a manually operable switch for selecting between the first and second voltages for the enable/disable signal and thereby for controlling the power delivered to said electrically powered data processing and storage components.

10. A personal computer system having a high speed system processor compatible with application programs and operating system software designed to execute on slower speed system processors, said personal computer system comprising:

a high speed microprocessor having a real and protected mode of operation coupled to a high speed data bus;

non-volatile memory electrically coupled to a slower speed data bus;

a bus controller for providing communications between the high speed data bus and the slower speed data bus;

volatile memory electrically responsive to the high speed data bus;

a memory controller electrically coupled to said volatile memory and said non-volatile memory, said memory controller regulating communications between said volatile memory and said high speed microprocessor; and

a pulse width modulation switching power supply for connection with an alternating current electrical main supply and for supplying direct current electrical power to said data processing and storage components for enabling operation thereof, said power supply comprising:

a controllable pulse width modulator for responding to the presence and absence of a low voltage direct current electrical signal by enabling and disabling the supply of electrical power to said data processing and storage components, and

a signal generator circuit operatively connected with said controllable component and with an alternating current electrical main supply for controllably deriving from the main supply a low voltage direct current signal for delivery to said controllable component, said signal generator circuit comprising a comparator, a first signal circuit for supplying to the comparator a low voltage direct current reference signal, and a controllable second signal circuit for controllably supplying to the comparator a low voltage enable/disable signal varying between a first voltage and a second, higher voltage, said comparator when supplied with said reference signal and said enable/disable signal having said first voltage delivering to said controllable component said first mentioned signal,

whereby a user of the microcomputer may control energization of the electrically powered data processing and storage components by controlling the application of said low voltage direct current signal from said signal generator circuit to said controllable component.

* * * * *

EXHIBIT 4

US005249741A

United States Patent [19]

Bistline et al.

[11] Patent Number: 5,249,741

[45] Date of Patent: Oct. 5, 1993

[54] AUTOMATIC FAN SPEED CONTROL

[75] Inventors: William R. Bistline; William C. Johnson; James M. Peterson, all of Austin, Tex.

[73] Assignee: International Business Machines Corporation, Armonk, N.Y.

[21] Appl. No.: 878,115

[22] Filed: May 4, 1992

[51] Int. Cl.[5] H02H 3/18

[52] U.S. Cl. 236/49.3; 62/259.3; 165/80.3; 236/DIG. 9; 361/694; 361/687; 454/184

[58] Field of Search ..... 236/49.3, DIG. 9; 361/384, 385; 454/184; 357/81; 363/141; 165/80.3, 80.4; 62/186, 259.3

[56] References Cited

U.S. PATENT DOCUMENTS

4,722,669	2/1988	Kundert	417/32
5,038,320	8/1991	Heath et al.	364/900

FOREIGN PATENT DOCUMENTS

0157507	10/1985	European Pat. Off.	361/384
58-99821	6/1983	Japan	361/384
58-129524	8/1983	Japan	361/384
02-54797	10/1990	Japan	361/384

OTHER PUBLICATIONS

IBM Technical Disclosure Bulletin, vol. 18, No. 6, Nov. 1975, "Cooling Control", R. A. Enrietto and G. Podhajsky, p. 1705.

IBM Technical Disclosure Bulletin, vol. 20, No. 8, Jan. 1978, "Computerized Control of Chilled Water System", p. 2981.

IBM Technical Disclosure Bulletin, vol. 32, No. 10A, Mar. 1990, "Variable Air Cooling for Computer And/Or Electronic Equipment", p. 196.

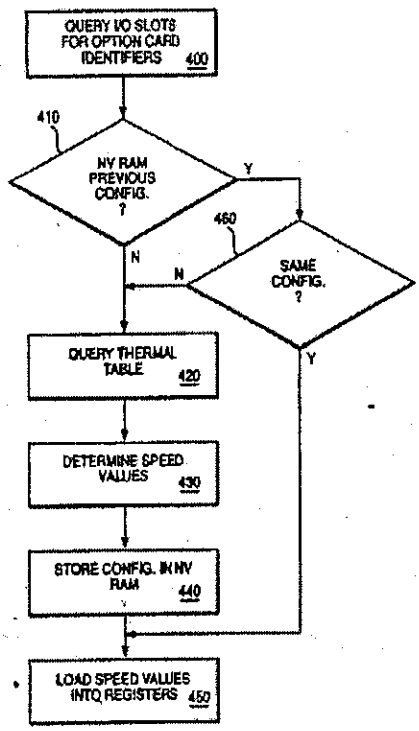
Primary Examiner—William E. Wayner

Attorney, Agent, or Firm—Paul S. Drake

[57] ABSTRACT

A method of cooling a computer having a plurality of components and at least one variable rate cooling unit including the steps of obtaining a cooling requirement for at least one of the components and varying the rate of at least one of the cooling units based on the obtained cooling requirements. In addition an apparatus for cooling a computer having a plurality of components, the apparatus including at least one variable rate cooling unit, an apparatus for obtaining a cooling requirement for at least one of the components, and an apparatus for varying the rate of at least one of the cooling units based on the obtained cooling requirements.

27 Claims, 3 Drawing Sheets



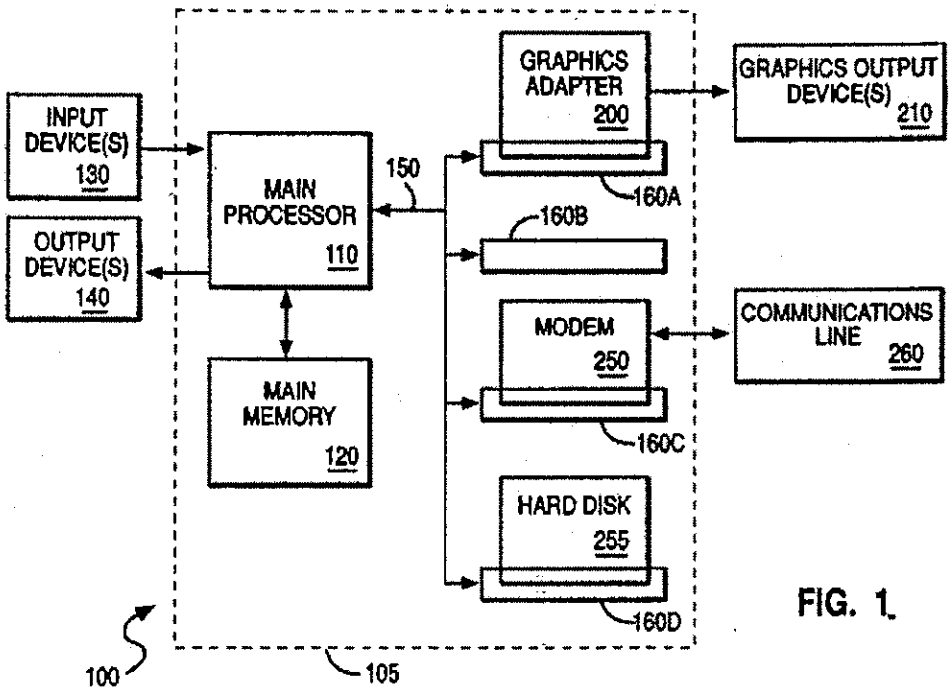


FIG. 1.

ID	DESCRIPTION	SPEED VALUE
B1	BAY 1	0100
B2	BAY 2	0011
F1	FAN 1	0111
F2	FAN 2	0111
00	HARD DISK	0010
01		
02	MODEM	0001
03		
98	GRAPHICS ADAPTER	0111
99		

FIG. 4

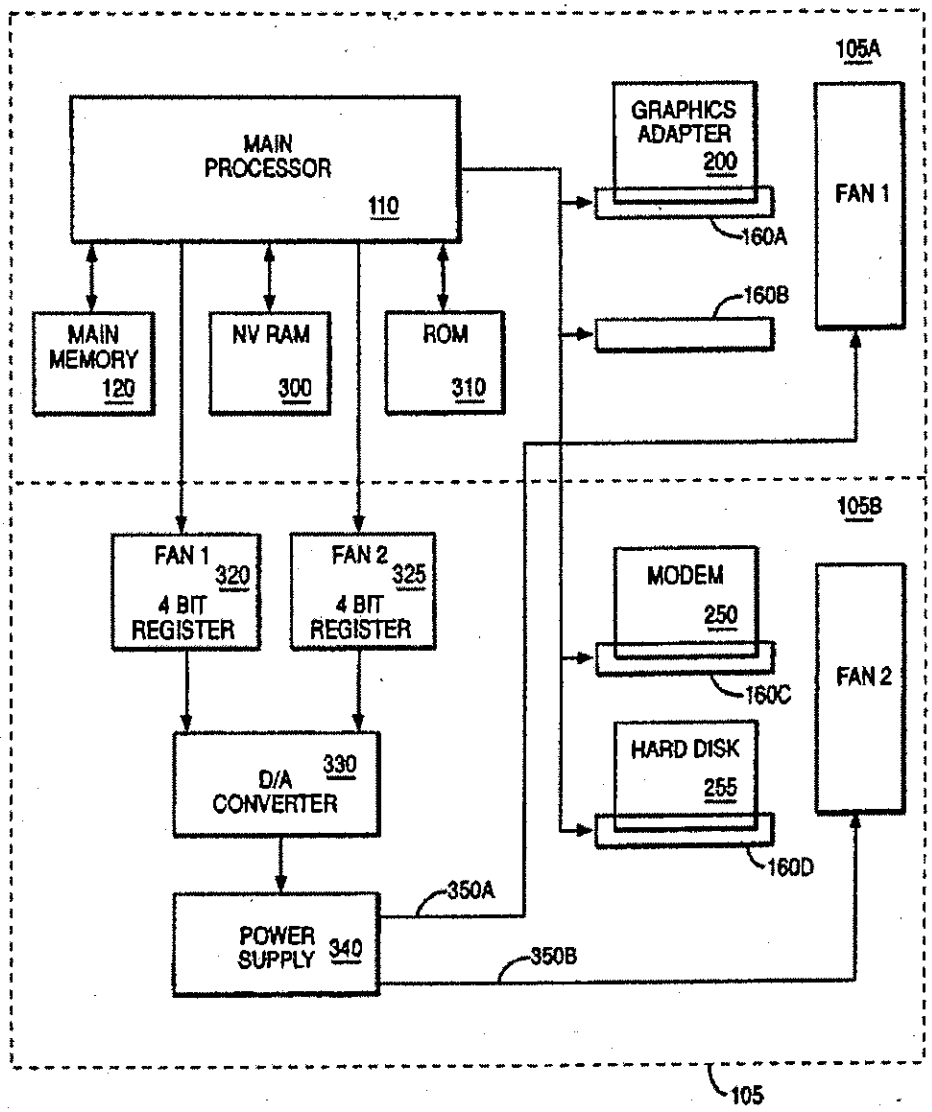

105A
MAIN PROCESSOR 110
GRAPHICS ADAPTER 200
160A
160B
FAN 1
MAIN MEMORY 120
NV RAM 300
ROM 310
105B
FAN 1 320 4 BIT REGISTER
FAN 2 325 4 BIT REGISTER
MODEM 250
160C
FAN 2
HARD DISK 255
160D
D/A CONVERTER 330
350A
POWER SUPPLY 340
350B
105

FIG. 2

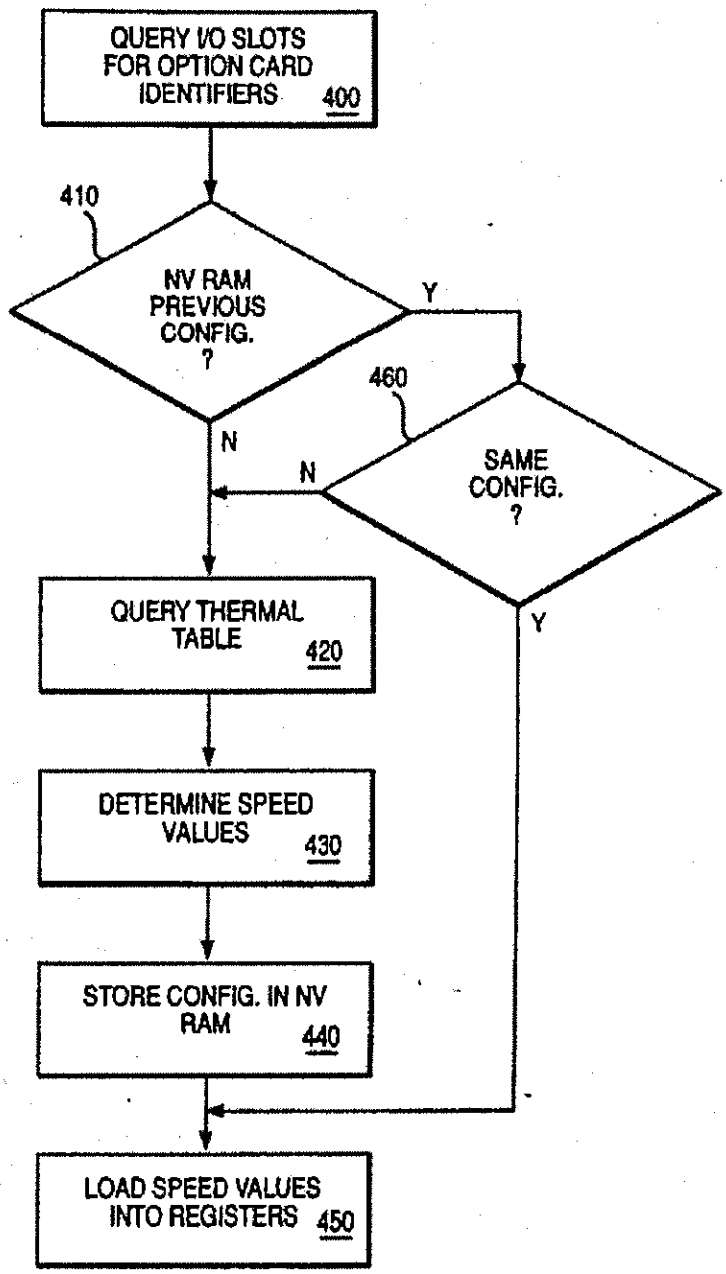

QUERY I/O SLOTS FOR OPTION CARD IDENTIFIERS 400
410
NV RAM PREVIOUS CONFIG. ?
Y
N
460
SAME CONFIG. ?
N
Y
QUERY THERMAL TABLE 420
DETERMINE SPEED VALUES 430
STORE CONFIG. IN NV RAM 440
LOAD SPEED VALUES INTO REGISTERS 450

FIG. 3

AUTOMATIC FAN SPEED CONTROL

DESCRIPTION

1. Technical Field

The present invention relates to computer system cooling and more particularly to automatic speed control of fans used for cooling a computer system.

2. Background Art

Computer systems include electronic equipment that produces heat when used. The amount of heat produced varies depending upon the type of equipment being utilized. For example, a high speed graphics adapter may produce a large amount of heat whereas a slow speed modem may produce a small amount of heat. If this heat is allowed to build up inside the computer, the electronic equipment may start producing errors and the lifetime of the electronic equipment may be shortened substantially. As a result, computer systems often utilize one or more fans to provide cooling of the electronic equipment. Often these fans are preset to a predetermined speed at the factory to provide a fixed amount of cooling.

However, fans produce a large amount of acoustic noise that is undesirable and may decrease the efficiency of the user utilizing the equipment or of others within the work environment. As a result, many computer systems now contain a variable speed fan. The speed of the fan is varied to minimize the noise generated by the fan while providing adequate cooling for the electronic equipment.

Various types of variable speed fan systems are utilized in the art. One common approach is to utilize a thermistor within the fan hardware to detect changing air temperature inside the computer and then vary the fan speed accordingly. An example of this is provided in "Variable Air Cooling for Computer And/Or Electronic Equipment" in IBM Technical Disclosure Bulletin Vol. 32, No. 10A, March 1990. The difficulty with using a thermistor is that the location of the thermistor may impact the accuracy of the temperature reading, the thermistor will detect a temperature rise some period of time after the temperature has already risen, and thermistors tend to degrade in performance over time.

Another popular approach for varying the speed of a fan is to vary the speed based upon the amount of current drawn from the computer power supply. As more current is drawn from the power supply more heat is generally generated by the computer system and greater cooling is required. However, this approach has several drawbacks. It is expensive to implement and if there are particularly hot components such as a graphics adapter located within the system, a greater amount of cooling may be required to keep that particular component cool.

The prior art approaches to varying fan speed become less adequate when multiple fans are used. In addition as computer systems become more complex and modular, there may be a greater number of hot spots located in various areas within the computer system that may be cooled by different fans.

DISCLOSURE OF THE INVENTION

The present invention includes a method of cooling a computer having a plurality of components and at least one variable rate cooling unit including the steps of obtaining a cooling requirement for at least one of the components and varying the rate of at least one of the cooling units based on the obtained cooling requirements. In addition, the present invention includes an apparatus for cooling a computer having a plurality of components, the apparatus including at least one variable rate cooling unit, an apparatus for obtaining a cooling requirement for at least one of the components, and an apparatus for varying the rate of at least one of the cooling units based on the obtained cooling requirements.

A further understanding of the nature and advantages of the present invention may be realized by reference to the remaining portions of the specification and the drawings.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a block diagram of a typical digital computer utilized by a preferred embodiment of the invention;

FIG. 2 is a detailed block diagram of the computer system of FIG. 1 illustrating a preferred cooling system used to cool the system;

FIG. 3 is a flowchart illustrating how the speed values are determined for each fan; and

FIG. 4 is an example of a thermal table stored on the hard disk.

BEST MODE FOR CARRYING OUT THE INVENTION

FIG. 1 is a block diagram of a typical digital computer 100 utilized by a preferred embodiment of the invention. One computer that could potentially utilize the present invention is the IBM RISC System/6000 (trademark of International Business Machines Corporation). The computer includes a computer box 105 which encloses a main processor or central processing unit (CPU) 110 coupled to a memory 120. The main processor may communicate with input device(s) 130 and output device(s) 140 located outside the computer box 105. Main processor 110 may include a single processor or multiple processors. Memory 120 may generally be expanded in amount by adding more memory cards. Input device(s) 130 may include a keyboard, mouse, tablet or other types of input devices. Output device(s) 140 may include a text monitor, plotter or other types of output devices.

The main processor may also be coupled within computer box 105 to multiple input/output (I/O) slots 160A, 160B, 160C and 160D via bus 150. Bus 150 may be a standard bus such as the Micro Channel (trademark of International Business Machines Corporation). More or fewer I/O slots may be provided depending upon the computer system. The multiple I/O slots provide the capability to insert various option cards into the computer system which would then communicate with the main processor via bus 150.

An example of an option card inserted into the I/O slots would be a graphics adapter 200 which would be attached to a graphics output device such as a graphics display or plotter 210. Graphics adapter 200 would then receive instructions regarding graphics from main processor 110 on bus 150. The graphics adapter then executes those instructions and provides the results to the graphics output device 210 thereby rendering the desired graphics output from the main processor.

Another example of an option card inserted into an I/O slot would be a modem 250 which would then be attached to a communications line 260 such as a tele-

phone line. Modem 250 would then receive instructions from main processor 110 to initiate communications over the communication lines 260 to a similarly configured computer system. The modem would then establish the desired communications link and provide communication from the main processor to the other computer system as well as provide communication from the other system to the main processor. Modem 250 may also receive requests from other computer systems over communications line 260 to initiate communications with main processor 110.

Many other types of option cards may be utilized in the I/O slots 160A–160D such as hard disk 255 or memory cards. Hard disk 255 represents a hard disk adapter plugged into the I/O slot in combination with a hard disk drive located next to the adapter. In general any of the option cards may be inserted into any of the available I/O slots. In addition, some option cards may be hotter than others and greater cooling may be needed for that area of the computer system. Cooling is generally required for all electronic devices included in the computer box 105.

FIG. 2 is a detailed block diagram of the computer system of FIG. 1 illustrating the cooling system used to cool the computer system. For illustrative purposes, the computer box 105 includes two bays or open areas 105A and 105B. In this example, each bay is thermally separated from the other bay thereby requiring separate fans for cooling. Bay 105A includes fan 1 and bay 105B includes fan 2, each fan providing the necessary cooling for the components included in that bay. Fan 1 cools the main processor 110, main memory 120, NVRAM (non-volatile RAM which is generally powered by a battery when the computer is turned off) 300, ROM 310, I/O slot 160A with the graphics adapter 200 and any component inserted into I/O slot 160B. Fan 2 cools registers 320 and 325, D/A (digital to analog) converter 330, power supply 340, I/O slot 160C with modem 250 and I/O slot 160D with hard disk 255.

Each fan's speed is varied based upon the power provided to the fan by the power supply. That is, the speed of fan 1 is controlled by the voltage on power line 350A and the speed of fan 2 is controlled by the voltage on power line 350B. For example a voltage of 16 volts D.C. will run a fan at full speed whereas a voltage of 8 volts D.C. will run the same fan at half speed. In the preferred embodiment, the speed for each of the fans is stored in registers 320 and 325. Each register contains a 4 bit number, called a speed value, which designates the speed for the respective fan. In alternative embodiments, more or fewer bits may be used to designate the speed value. The speed value stored in each of the registers is converted to an analog value for input to the power supply 340. The power supply then provides the designated amount of voltage to each of the respective fans thereby controlling the amount of cooling performed by that fan.

FIG. 3 is a flowchart illustrating how the speed values for each fan are determined in the preferred embodiment. FIG. 3 will be described with reference to FIG. 2 for illustrative purposes. When the computer system is booted or powered up by the user, an initial program load program stored in ROM (read-only memory) 310 is used to initiate the processing of the computer. In the preferred embodiment IPLROS (initial program load read only storage) stored on ROM is utilized to boot the computer system. IPLROS is similar to BIOS used in many personal computers today. In the preferred embodiment, IPLROS is also used to determine the speed value for each of the fans during the computer boot. This is due to the fact that option cards are generally moved, added or removed while the system is turned off.

In step 400, The IPLROS may query the I/O slots to determine what option cards may be in the I/O slots. As a result of this polling an option card identifier is obtained for each of the I/O slots containing an option card. In the preferred embodiment, the option card identifier is a two byte identifier and may be obtained as taught in Heath et al. (U.S. Pat. No. 5,038,320). In step 410, the NVRAM 300 is then queried to determine if a previous configuration for the computer is stored there. If not, as will be explained below, then this is probably the first time the computer system has been booted and the speed value for each of the fans needs to be determined.

In step 420, a thermal table is queried to determine the base cooling requirements for each bay of the computer system and the cooling requirements for each of the components. In the preferred embodiment, the thermal table is stored on the hard disk when the computer system is manufactured and the thermal table may be updated by updates in the operating system or the like. In alternative embodiments, the thermal table may be stored in NVRAM or in ROM and may be updated by updates in the operating system. The updates to the thermal table may be stored in NVRAM or on the hard disk. In the preferred embodiment, the thermal table provides a 4 bit value representing the minimum cooling requirement for each bay of a base configured system including the memory and main processor. The thermal table also provides a 4 bit value representing the cooling requirement for each possible option card. The thermal table may also provide a 4 bit value representing a minimum speed for each fan regardless of what the thermal requirements may be for the respective bay (fans have generally have minimum start up speeds). In step 420, if there is no corresponding entry in the thermal table for an option card then the IPLROS will designate a default value, preferable full speed, for the unknown option card. In an alternative embodiment, the user may designate a speed value for the unknown option card.

In step 430, the speed value for each of the fans is then determined. In the preferred embodiment, the base cooling requirement for each bay and the cooling requirements for each of the components in the bay are accumulated to provide the speed value. An example of this is described with reference to FIG. 4 below. Once the speed values are determined then in step 440, the system configuration, including the location of the option cards and the speed values for each fan are stored in NVRAM for future reference. Then in step 450, the speed values are stored in four bit registers which value is then converted by a digital-to-analog converter into an analog value. This analog value is then provided to the power supply for each fan thereby setting the speed of that fan.

If in step 410 a previous configuration was stored in NVRAM, then in step 460 the current configuration is compared to the previous configuration including location of the various options cards to determine if the configuration has changed. If the configuration has changed, then the speed values need to be determined and processing continues to step 420. If the configuration is the same, then the previously determined speed

values may be used again and processing continues to step 450.

FIG. 4 is an example of a thermal table stored on the hard disk. In the present example B1 and B2 are used to represent the base cooling requirement or speed values for each of the bays. According to the table, B1 or bay 1 would require a speed value of at least 0100 to cool the main processor and memory and B2 or bay 2 would require a speed value of at least 0011 to cool the power supply. In addition F1 and F2 or fan 1 and fan 2 have a minimum speed of 0111.

In the preferred embodiment, the speed values for each bay is determined by accumulating the base cooling requirement or speed value with the cooling requirements or speed values for each of the option cards. According to the table bay 1 would have a speed value of 0100 for the base bay plus 0111 for the graphics adapter resulting in a total speed value of 1011 for fan 1 or about 11 volts for a 16 volt fan. Bay 2 would have a speed value of 0011 for the base bay plus 0010 for the hard disk and 0001 for the modem resulting in a total speed value of 0110 for fan 2 or about 6 volts for a 16 volt fan. However, 0110 is below the minimum speed for fan 2 (0111 as discussed above) so the speed value for fan 2 is set to 0111 or about 7 volts for a 16 volt fan.

If the two bays were not totally thermally separated, then a weighted average of the cooling requirements for each bay may be used to determine the speed values for each fan. In addition, if the total cooling requirement for any bay exceeds the capability of the fans, then the user may be notified by the system to move the option cards to a compliant condition.

Although the present invention has been fully described above with reference to specific embodiments, other alternative embodiments will be apparent to those of ordinary skill in the art. For example, other portions of the computer system besides the I/O slots, such as the CPU chip when the CPU chip may be replaced, may be queried to determine the appropriate speed values for the respective fans. In addition, other cooling technologies such as water cooling may be utilized with the present invention. These alternative cooling technologies may preferably be variable speed for acoustic reasons or possibly to lessen wear on cooling system components such as a water pump. Therefore, the above description should not be taken as limiting the scope of the present invention which is defined by the appended claims.

What is claimed is:

1. A method of cooling a computer having a plurality of components and at least one variable rate a fan cooling unit comprising the steps of:

a) obtaining a predetermined cooling requirement for at least one of said components; and

b) varying the rate of at least one of said cooling units based on the obtained cooling requirements.

2. The method of claim 1 wherein the step of obtaining includes the steps of:

a) querying said components for component identifiers; and

b) using said identifiers to obtain cooling requirements for said components.

3. The method of claim 2 wherein said step of varying includes the steps of:

a) determining for at least one of said cooling units the cooling requirements for components cooled by said cooling unit; and

b) setting the rate of at least one of said cooling units based on the determined cooling requirements.

4. The method of claim 3 wherein said step of determining includes accumulating for each cooling unit the cooling requirements for components cooled by said cooling unit.

5. The method of claim 3 wherein said step of determining includes determining for each cooling unit the weighted average of the cooling requirements for components cooled by said cooling unit.

6. An apparatus for cooling a computer having a plurality of components, said apparatus comprising:

a) at least one variable rate cooling unit;

b) means for obtaining a predetermined cooling requirement for at least one of said components; and

c) means for varying the rate of at least one of said cooling units based on the obtained cooling requirements.

7. The apparatus of claim 6 wherein the means for obtaining includes:

a) means for querying said components for component identifiers; and

b) using said identifiers to obtain cooling requirements for said components.

8. The apparatus of claim 7 wherein said means for varying includes:

a) means for determining for at least one of said cooling units the cooling requirements for components cooled by said cooling unit; and

b) means for setting the rate of at least one of said cooling units based on the determined cooling requirements.

9. The apparatus of claim 8 wherein said means for determining includes means for accumulating for each cooling unit the cooling requirements for components cooled by said cooling unit.

10. The apparatus of claim 8 wherein said means for determining includes means for determining for each cooling unit the weighted average of the cooling requirements for components cooled by said cooling unit.

11. A computer system comprising:

a) a plurality of components including a main processor and a memory; and

b) a cooling system for cooling said plurality of components, said cooling system including:

i) at least one variable rate cooling unit;

ii) means for obtaining predetermined cooling requirements for at least one of said components; and

iii) means for varying the rate of at least one of said cooling units based on the obtained cooling requirements.

12. A computer system comprising:

a) a plurality of components including a main processor and a memory; and

b) a cooling system for cooling said plurality of components, said cooling system including:

i) at least one variable rate cooling unit;

ii) means for obtaining predetermined cooling requirements for at least one of said components, said means for obtaining including:

means for querying said components for component identifiers; and

using said identifiers to obtain cooling requirements for said components; and

iii) means for varying the rate of at least one of said cooling units based on the obtained cooling requirements.

13. The computer system of claim 12 wherein said means for varying includes:

a) means for determining for at least one of said cooling units the cooling requirements for components cooled by said cooling unit: and

b) means for setting the rate of at least one of said cooling units based on the determined cooling requirements.

14. The computer system of claim 13 wherein said means for determining includes means for accumulating for each cooling unit the cooling requirements for components cooled by said cooling unit.

15. The computer system of claim 13 wherein said means for determining includes means for determining for each cooling unit the weighted average of the cooling requirements for components cooled by said cooling unit.

16. The computer system of claim 11 wherein the means for obtaining includes:

a) means for querying said components for component identifiers; and

b) using said identifiers to obtain cooling requirements for said components.

17. The computer system of claim 16 wherein said means for varying includes:

a) means for determining for at least one of said cooling units the cooling requirements for components cooled by said cooling unit; and

b) means for setting the rate of at least one of said cooling units based on the determined cooling requirements.

18. The computer system of claim 17 wherein said means for determining includes means for accumulating for each cooling unit the cooling requirements for components cooled by said cooling unit.

19. The computer system of claim 17 wherein said means for determining includes means for determining for each cooling unit the weighted average of the cooling requirements for components cooled by said cooling unit.

20. A method of cooling a computer having a plurality of components and at least one variable rate cooling unit comprising the steps of:

a) obtaining a cooling requirement for at least one of said components including the steps of:

i) querying said components for component identifiers; and

ii) using said identifiers to obtain cooling requirements for said components; and

b) varying the rate of at least one of said cooling units based on the obtained cooling requirements.

21. The method of claim 20 wherein said step of varying includes the steps of:

a) determining for at least one of said cooling units the cooling requirements for components cooled by said cooling unit; and

b) setting the rate of at least one of said cooling units based on the determined cooling requirements.

22. The method of claim 12 wherein said step of determining includes accumulating for each cooling unit the cooling requirements for components cooled by said cooling unit.

23. The method of claim 21 wherein said step of determining includes determining for each cooling unit the weighted average of the cooling requirements for components cooled by said cooling unit.

24. An apparatus for cooling a computer having a plurality of components, said apparatus comprising:

a) at least one variable rate cooling unit;

b) means for obtaining a cooling requirement for at least one of said components, said means for obtaining including:

i) means for querying said components for component identifiers; and

ii) using said identifiers to obtain cooling requirements for said components; and

c) means for varying the rate of at least one of said cooling units based on the obtained cooling requirements.

25. The apparatus of claim 24 wherein said means for varying includes:

a) means for determining for at least one of said cooling units the cooling requirements for components cooled by said cooling unit; and

b) means for setting the rate of at least one of said cooling units based on the determined cooling requirements.

26. The apparatus of claim 25 wherein said means for determining includes means for accumulating for each cooling unit the cooling requirements for components cooled by said cooling unit.

27. The apparatus of claim 25 wherein said means for determining includes means for determining for each cooling unit the weighted average of the cooling requirements for components cooled by said cooling unit.

* * * * *

EXHIBIT 5

US005371852A

United States Patent [19]

Attanasio et al.

[11] Patent Number: 5,371,852

[45] Date of Patent: Dec. 6, 1994

[54] METHOD AND APPARATUS FOR MAKING A CLUSTER OF COMPUTERS APPEAR AS A SINGLE HOST ON A NETWORK

[75] Inventors: Clement R. Attanasio, Peekskill; Stephen E. Smith, Mahopac, both of N.Y.

[73] Assignee: International Business Machines Corporation, Armonk, N.Y.

[21] Appl. No.: 960,742

[22] Filed: Oct. 14, 1992

[51] Int. Cl.[5] G06F 13/00

[52] U.S. Cl. 395/200; 370/85.13

[58] Field of Search 395/200, 500; 370/60, 370/92, 93, 94.1, 54, 85.13, 85.1, 85.6, 85.8, 60.1, 110.1, 85.11, 95.1; 364/284.4, 242.94, 284.3, 284

[56] References Cited

U.S. PATENT DOCUMENTS

4,276,643	6/1981	Laprie et al.	371/8
4,667,287	5/1987	Allen et al.	364/200
4,719,621	1/1988	May	370/85
4,958,273	9/1990	Anderson et al.	364/200
5,023,780	6/1991	Brearley	364/200
5,079,765	1/1992	Nakamura	370/85.13
5,088,032	2/1992	Bosack	395/200
5,093,920	3/1992	Agrawal et al.	395/800
5,109,515	4/1992	Laggis et al.	395/725
5,125,081	6/1992	Chiba	395/325
5,166,931	11/1992	Riddle	370/94.1
5,185,860	2/1993	Wu	395/200
5,224,205	6/1993	Dinkin et al.	395/200

FOREIGN PATENT DOCUMENTS

1002342	1/1991	Belgium .
59-117842	7/1984	Japan .
63-193739	8/1988	Japan .

OTHER PUBLICATIONS

D. E. Comer, "Internetworking with TCP/IP, Principles, Protocols and Architecture", Prentice Hall, US, Chapter 7, pp. 91–97; Chapter 11, pp. 159–169; and Chapter 12, pp. 171–203.

"Network Services", Network Programming, Sun Microsystems, Inc., 2550 Garcia Ave., Mountain View, Calif., Chapter 1, pp. 3–30, May 1988.

OSF DCE Release 1.0 S3, DCE Release Notes, Open Software Foundation, 11 Cambridge Center, Cambridge, Mass., Mar. 25, 1991, pp. i–v, 1–1 thru B–2.

J. K. Ousterhout et al., "The Sprite Network Operating System", IEEE Computer, Feb. 1988, US, pp. 23–36.

D. R. Cheriton, "The V Distributed System", Communication of the ACM, Mar. 1988, vol. 31, No. 3, pp. 314–333.

A. Bhide et al., "A Highly Available Network File Server", USENIX Conference, Winter 1991, Dalas, Tex., p. 199.

S. J. Mullender et al., "Amoeba A distributed Operating System for the 1990s", IEEE Computer, May 1990, US, pp. 44–53.

L. Peterson et al., "The X-kernel: A Platform for Accessing Internet Resources", IEEE Computer, May 1990, US, pp. 23–33.

A. Litman, "The DUNIX Distributed Operating System", Operating Systems Review, ACM Press, NY, vol. 22, No. 1, Jan. 1988, pp. 42–51.

Primary Examiner—Dale M. Shaw

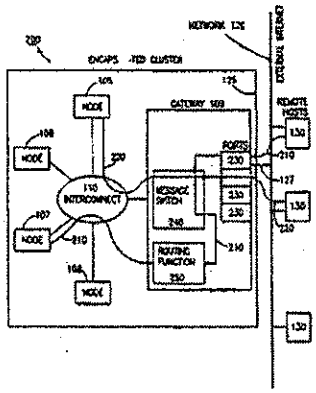
Assistant Examiner—Moustafa M. Meky

Attorney, Agent, or Firm—Louis J. Percello

[57] ABSTRACT

The present invention provides a method and apparatus for enabling a cluster of computers to appear as a single computer to host computers outside the cluster. A host computer communicates only with a gateway to access destination nodes and processes within the cluster. The gateway has at least one message switch which processes incoming and outgoing port type messages crossing the cluster boundary. This processing comprises examining certain information on the message headers and then changing some of this header information either to route an incoming message to the proper computer node, port and process or to make an outgoing message appear as if originated at the gateway node. The message switch uses a table to match incoming messages to a particular routing function which can be run to perform the changes necessary to correctly route different kinds of messages.

35 Claims, 13 Drawing Sheets



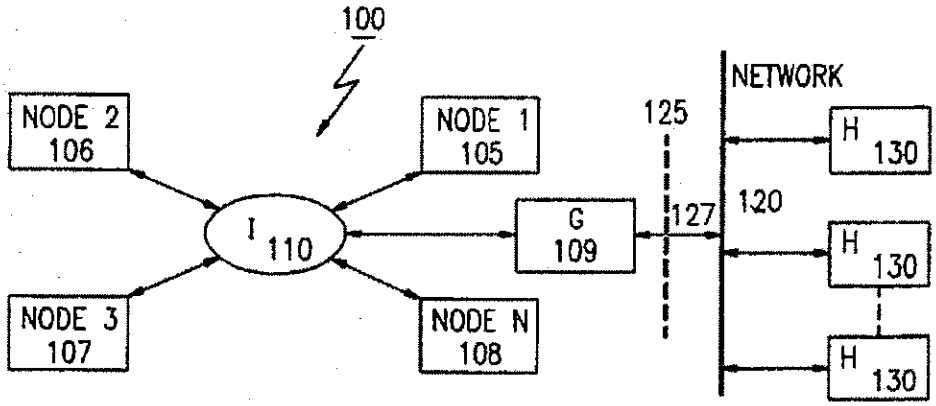


FIG. 1A
PRIOR ART

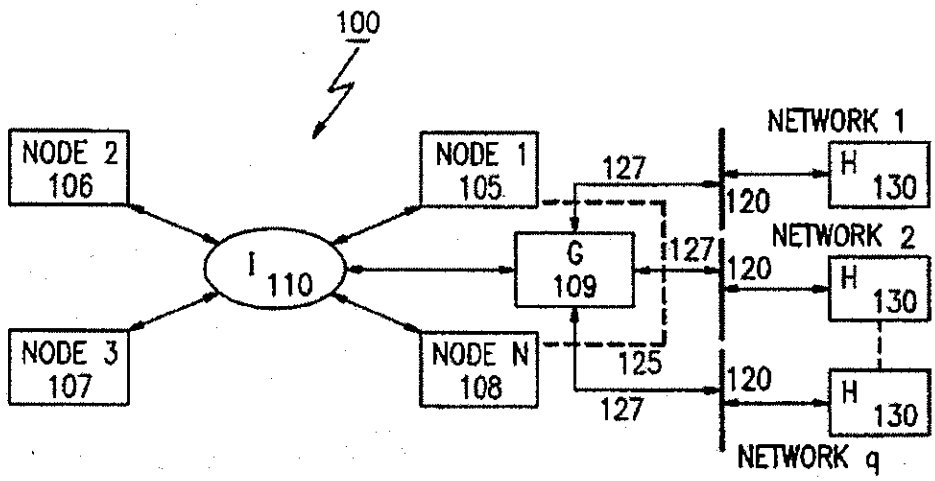


FIG. 1B
PRIOR ART

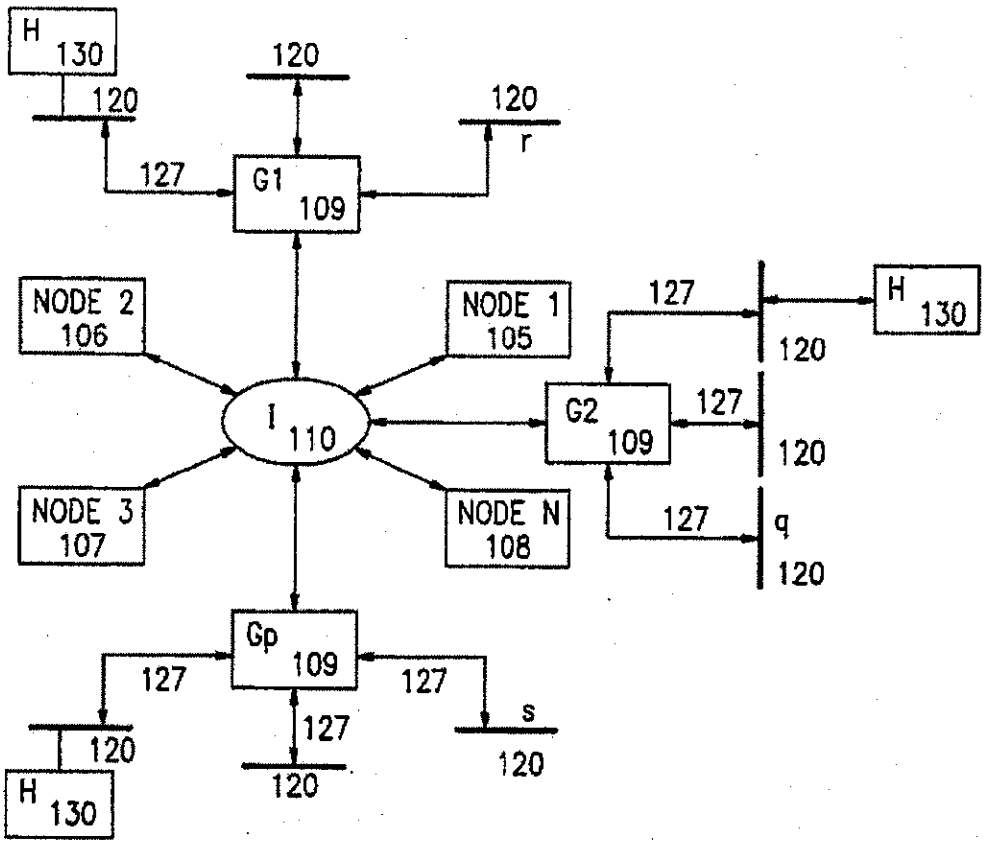


FIG. 1C
PRIOR ART

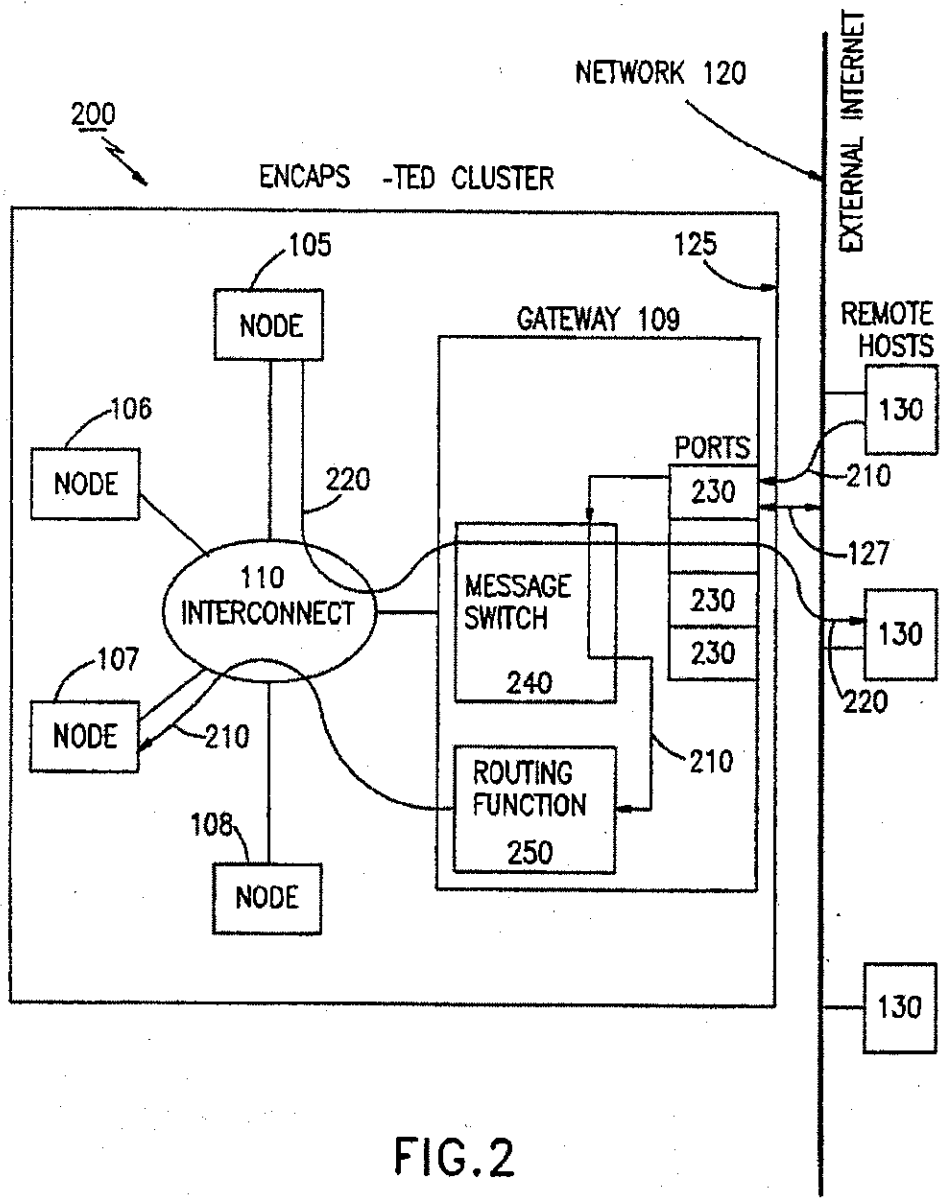

NETWORK 120
EXTERNAL INTERNET
200
ENCAPS -TED CLUSTER
105
125
NODE
GATEWAY 109
REMOTE HOSTS
130
106
NODE
220
PORTS
230
210
110
INTERCONNECT
MESSAGE SWITCH
240
127
230
230
130
107
NODE
210
220
210
ROUTING FUNCTION
250
108
NODE
130

FIG.2

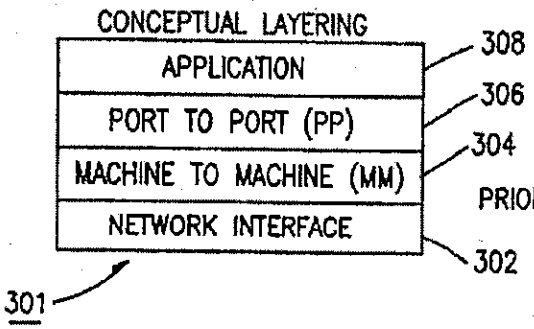
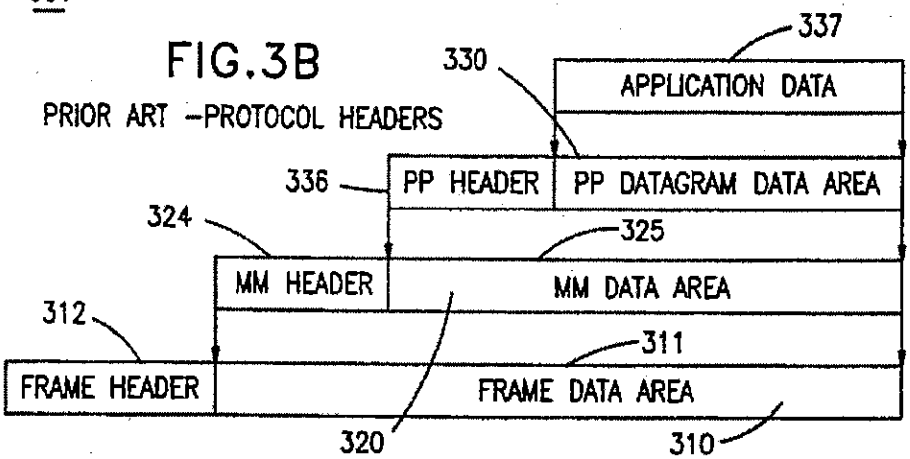


FIG.3A
PRIOR ART–PROTOCOL LAYERS

FIG.3B
PRIOR ART –PROTOCOL HEADERS



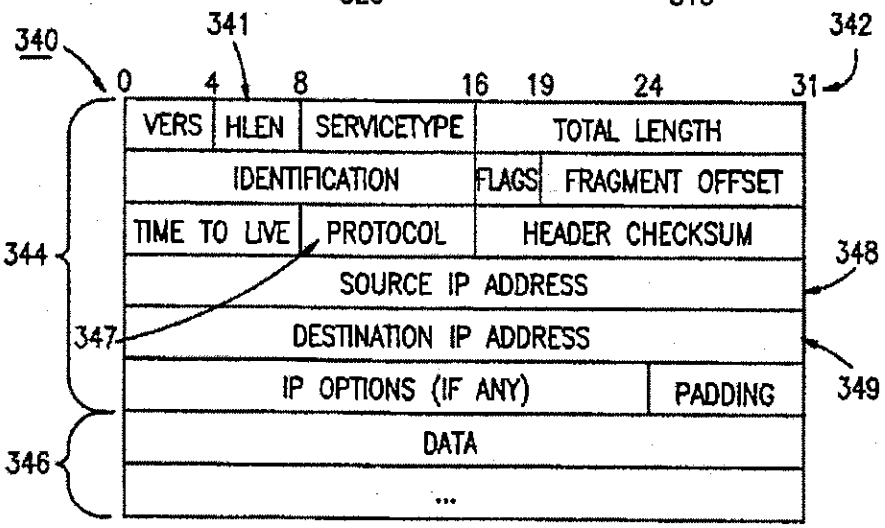


FIG.3C PRIOR ART–INTERNET PROTOCOL (IP) HEADER

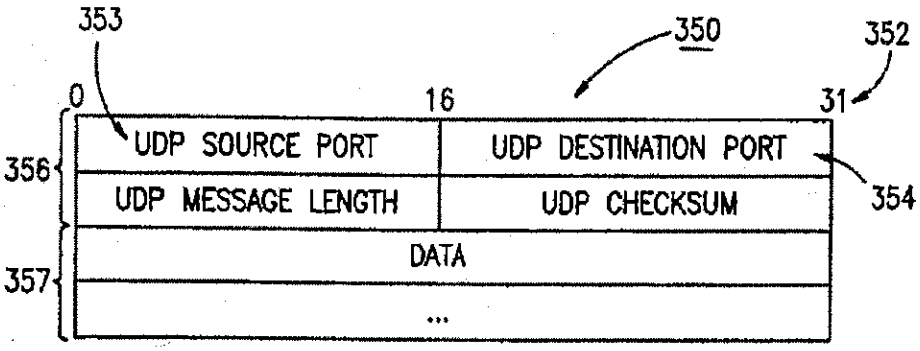


FIG.3D PRIOR ART—UDP DATAGRAM

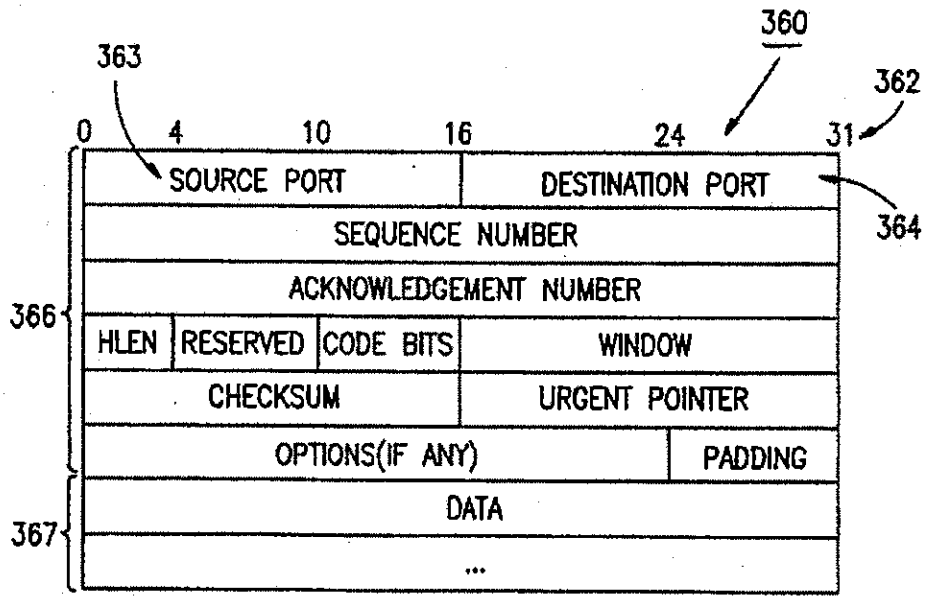


FIG.3E PRIOR ART—TCP DATAGRAM

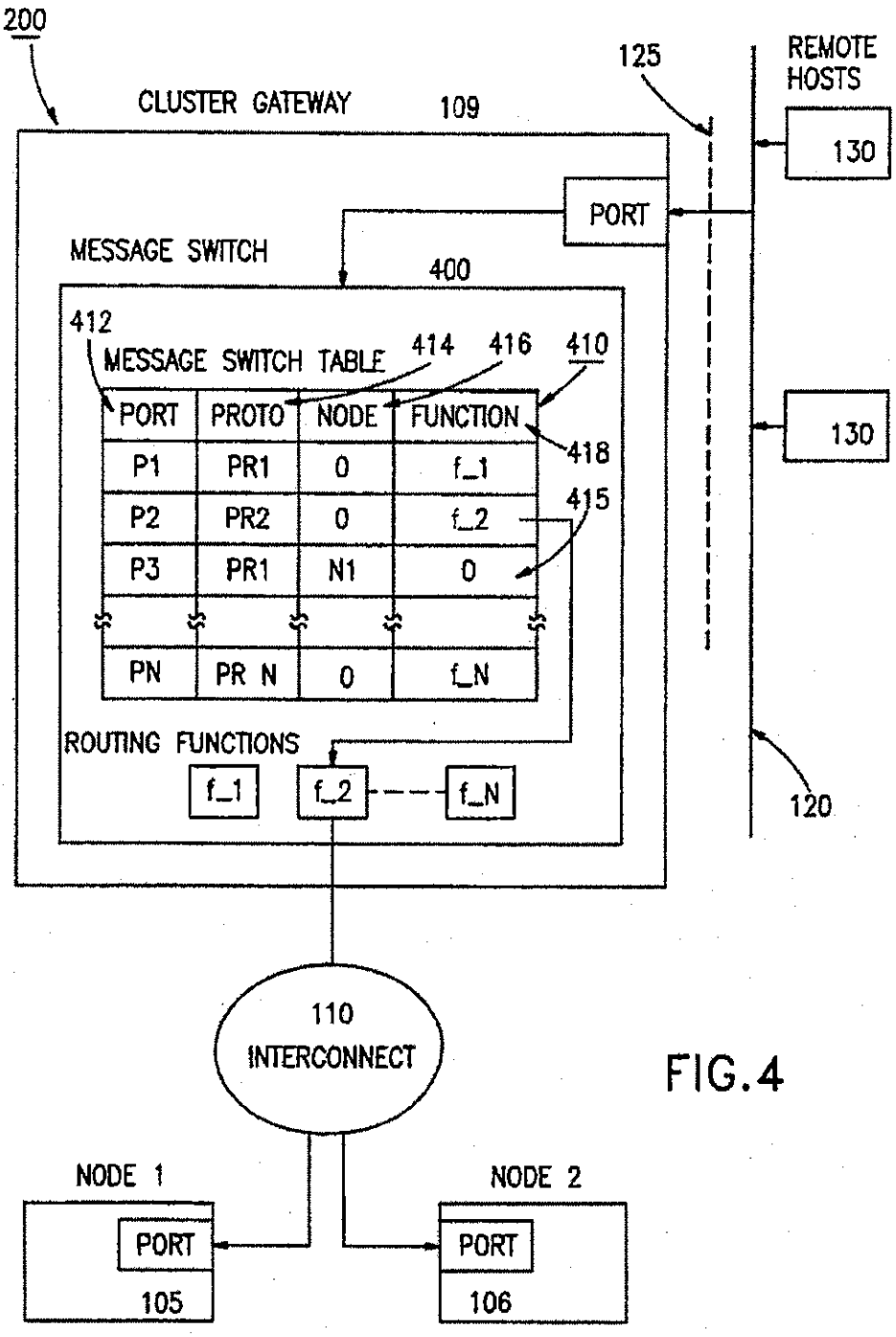

200
CLUSTER GATEWAY
109
125
REMOTE HOSTS
130
PORT
MESSAGE SWITCH
400
412
MESSAGE SWITCH TABLE
414
416
410
PORT
PROTO
NODE
FUNCTION
418
P1
PR1
0
f_1
P2
PR2
0
f_2
415
P3
PR1
N1
0
PN
PR N
0
f_N
130
ROUTING FUNCTIONS
f_1
f_2
f_N
120
110
INTERCONNECT
NODE 1
PORT
105
NODE 2
PORT
106

FIG.4

FIG.5A

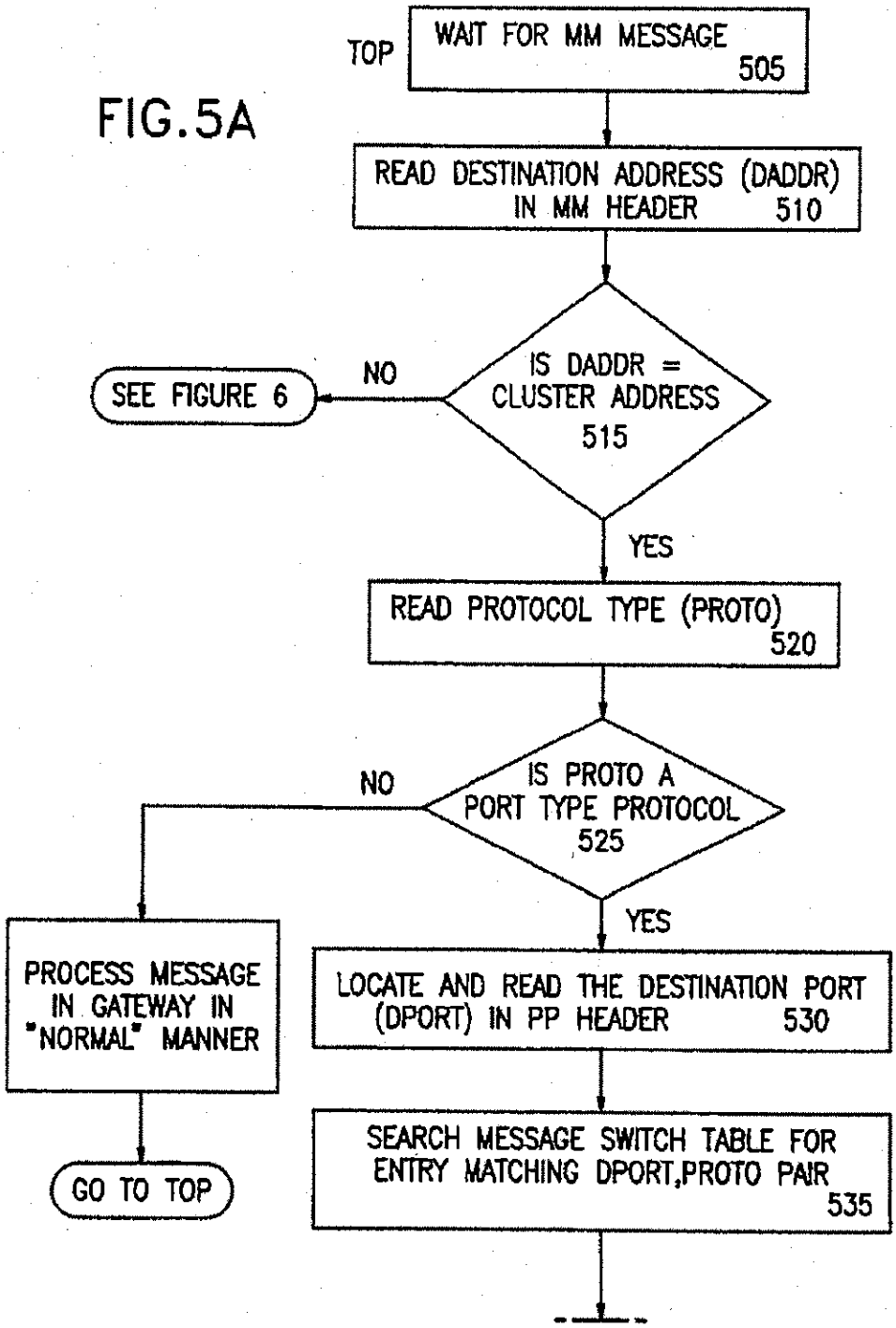

TOP
WAIT FOR MM MESSAGE 505
READ DESTINATION ADDRESS (DADDR) IN MM HEADER 510
IS DADDR = CLUSTER ADDRESS 515
NO
SEE FIGURE 6
YES
READ PROTOCOL TYPE (PROTO) 520
IS PROTO A PORT TYPE PROTOCOL 525
NO
YES
PROCESS MESSAGE IN GATEWAY IN "NORMAL" MANNER
GO TO TOP
LOCATE AND READ THE DESTINATION PORT (DPORT) IN PP HEADER 530
SEARCH MESSAGE SWITCH TABLE FOR ENTRY MATCHING DPORT,PROTO PAIR 535

FIG.5B

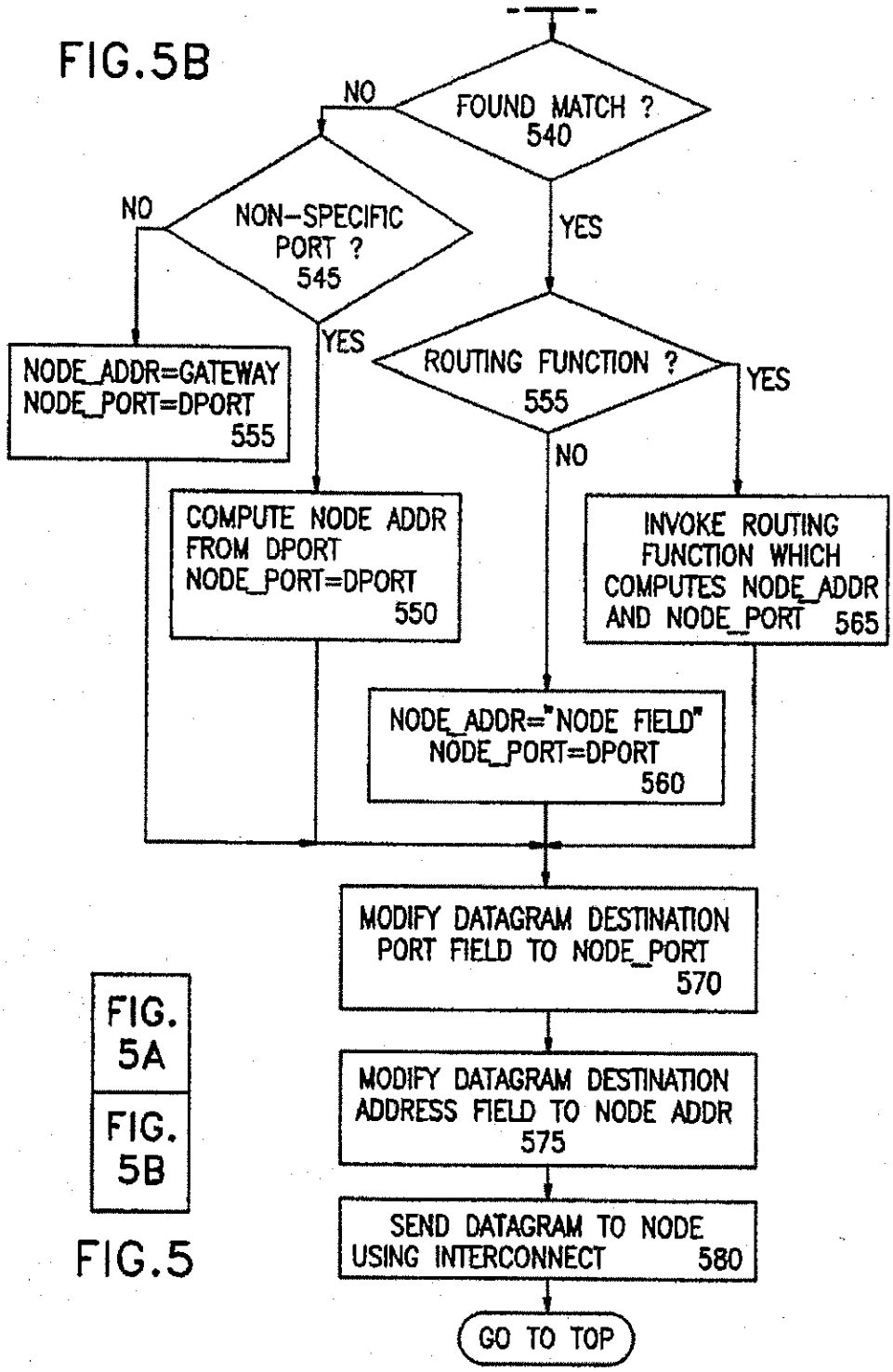
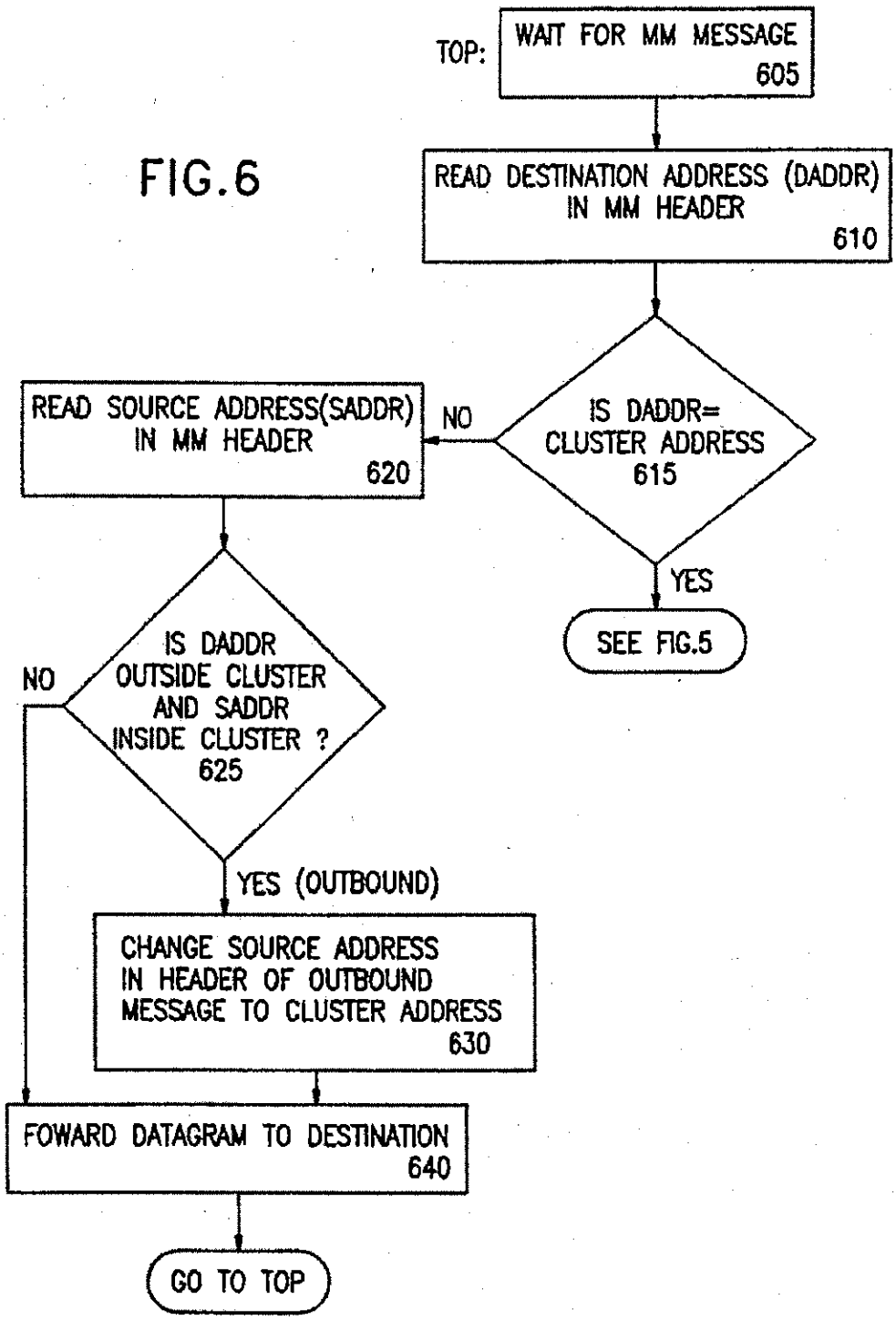


FIG. 5A

FIG. 5B

FIG.5

FIG.6


TOP:
WAIT FOR MM MESSAGE
605
READ DESTINATION ADDRESS (DADDR)
IN MM HEADER
610
IS DADDR=
CLUSTER ADDRESS
615
NO
YES
SEE FIG.5
READ SOURCE ADDRESS(SADDR)
IN MM HEADER
620
IS DADDR
OUTSIDE CLUSTER
AND SADDR
INSIDE CLUSTER ?
625
NO
YES (OUTBOUND)
CHANGE SOURCE ADDRESS
IN HEADER OF OUTBOUND
MESSAGE TO CLUSTER ADDRESS
630
FOWARD DATAGRAM TO DESTINATION
640
GO TO TOP

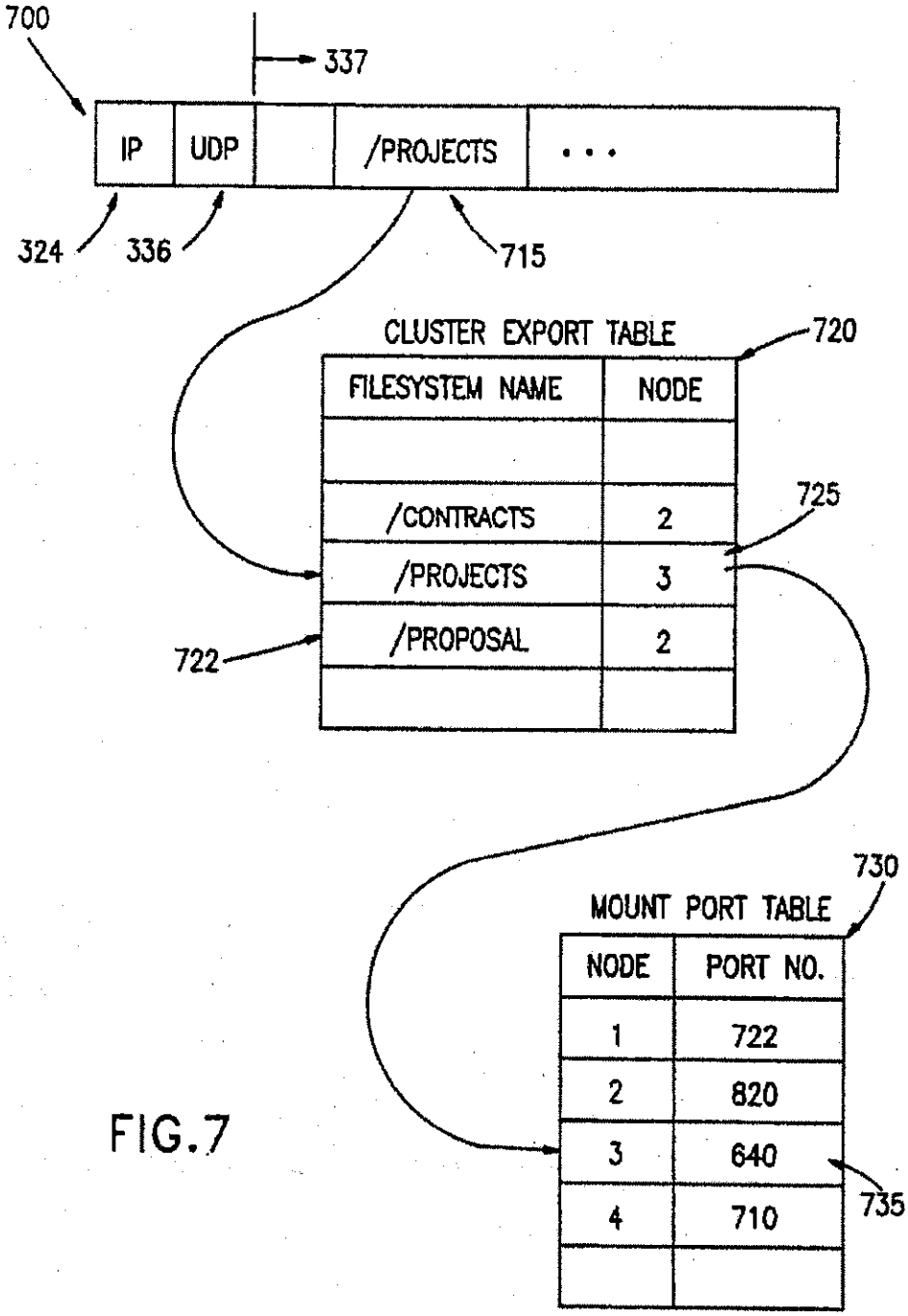

700
337
IP
UDP
/PROJECTS
• • •
324
336
715
CLUSTER EXPORT TABLE
720
FILESYSTEM NAME
NODE
/CONTRACTS
2
725
/PROJECTS
3
/PROPOSAL
2
722
730
MOUNT PORT TABLE
NODE
PORT NO.
1
722
2
820
3
640
4
710
735

FIG.7

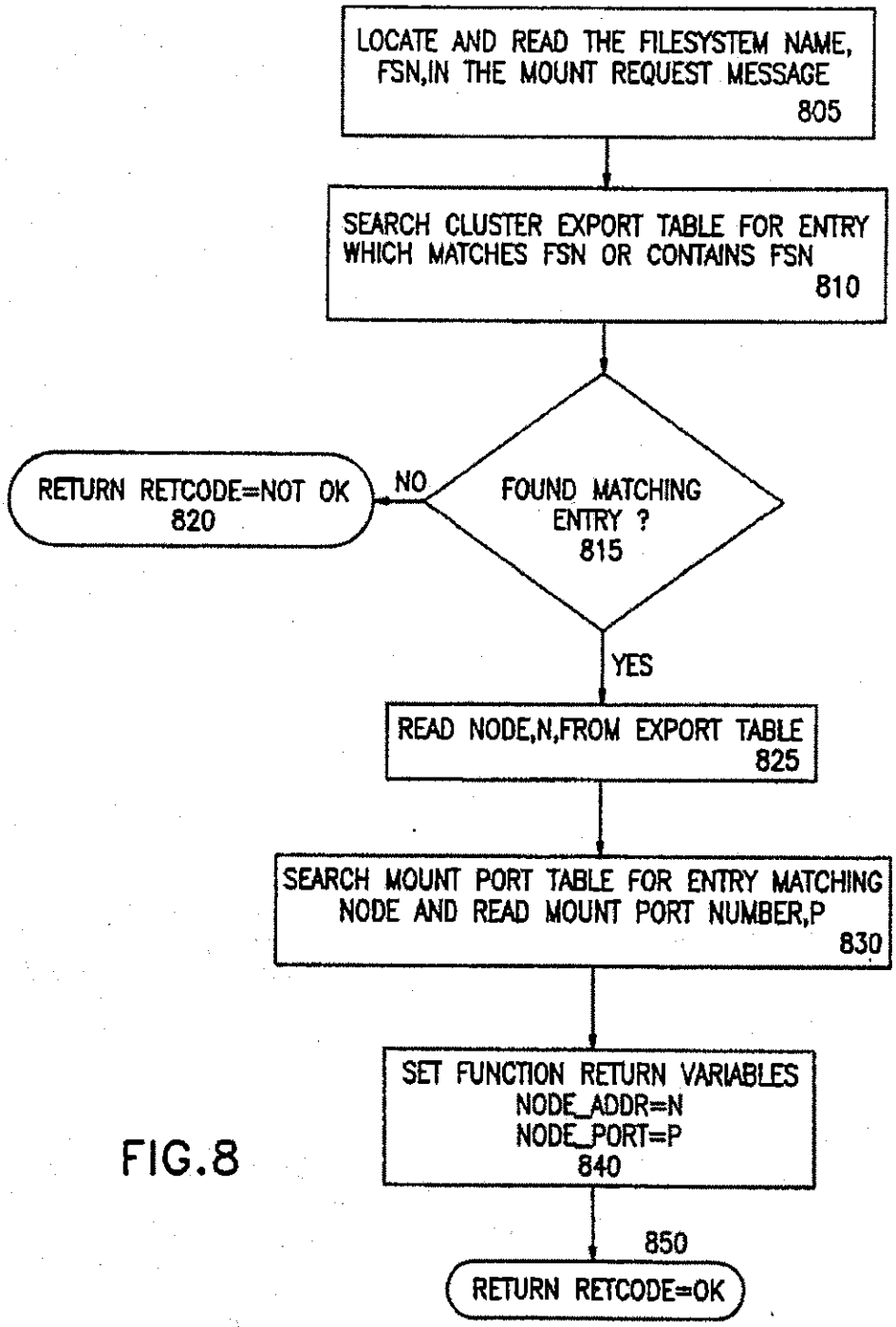

LOCATE AND READ THE FILESYSTEM NAME, FSN,IN THE MOUNT REQUEST MESSAGE
805
SEARCH CLUSTER EXPORT TABLE FOR ENTRY WHICH MATCHES FSN OR CONTAINS FSN
810
FOUND MATCHING ENTRY ?
815
NO
RETURN RETCODE=NOT OK
820
YES
READ NODE,N,FROM EXPORT TABLE
825
SEARCH MOUNT PORT TABLE FOR ENTRY MATCHING NODE AND READ MOUNT PORT NUMBER,P
830
SET FUNCTION RETURN VARIABLES
NODE_ADDR=N
NODE_PORT=P
840
850
RETURN RETCODE=OK

FIG.8

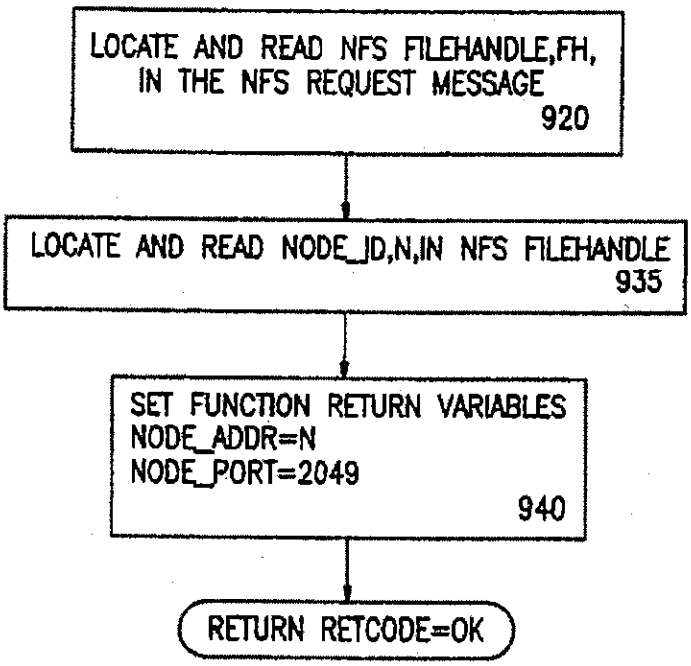

LOCATE AND READ NFS FILEHANDLE,FH, IN THE NFS REQUEST MESSAGE
920
LOCATE AND READ NODE_ID,N,IN NFS FILEHANDLE
935
SET FUNCTION RETURN VARIABLES
NODE_ADDR=N
NODE_PORT=2049
940
RETURN RETCODE=OK

FIG.9A

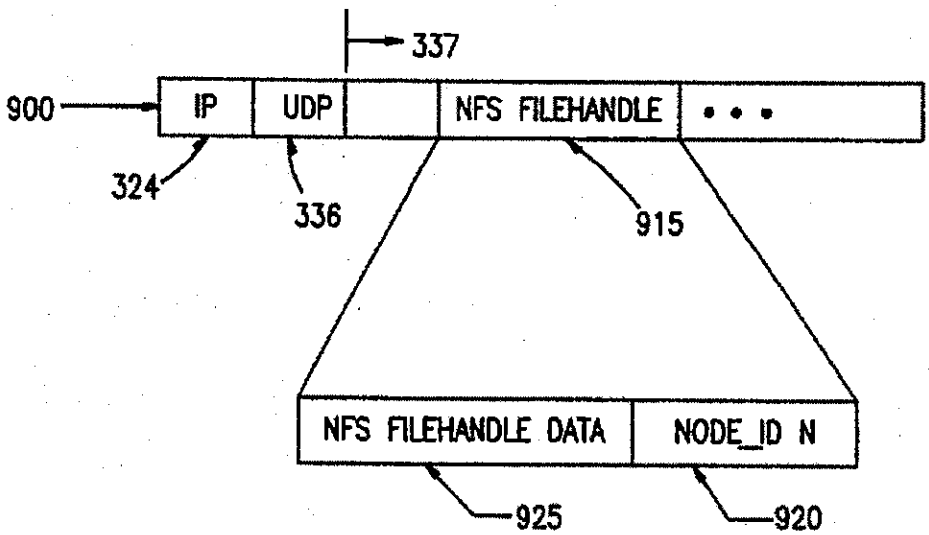

337
900
IP
UDP
NFS FILEHANDLE
• • •
324
336
915
NFS FILEHANDLE DATA
NODE_ID N
925
920

FIG.9B

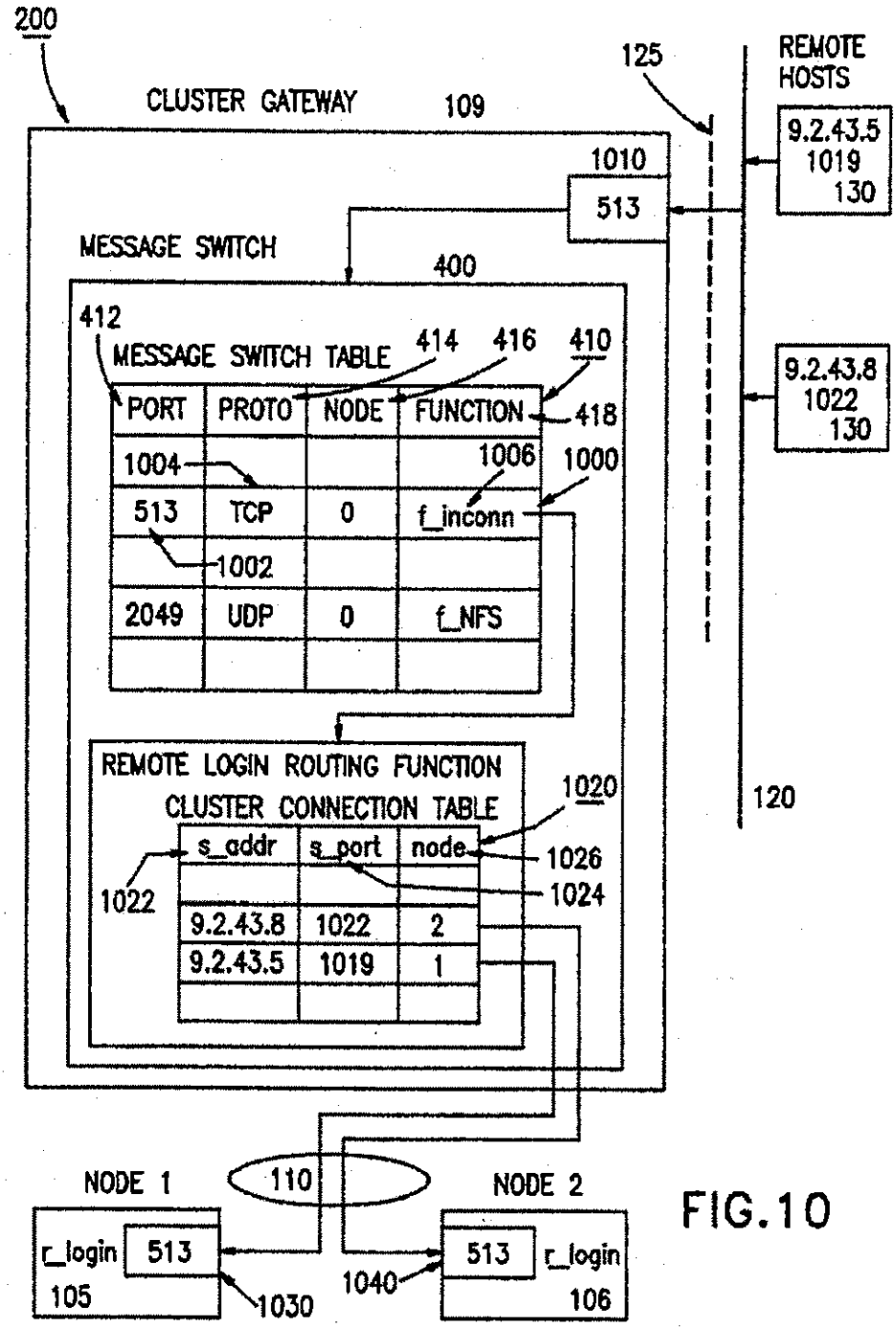

200
CLUSTER GATEWAY
109
125
REMOTE HOSTS
9.2.43.5
1019
130
1010
513
MESSAGE SWITCH
400
412
MESSAGE SWITCH TABLE
414
416
410
PORT
PROTO
NODE
FUNCTION
418
9.2.43.8
1022
130
1004
1006
1000
513
TCP
0
f_inconn
1002
2049
UDP
0
f_NFS
REMOTE LOGIN ROUTING FUNCTION
CLUSTER CONNECTION TABLE
1020
s_addr
s_port
node
1026
1024
1022
9.2.43.8
1022
2
9.2.43.5
1019
1
120
NODE 1
110
NODE 2
r_login
513
105
1030
1040
513
r_login
106

FIG.10

METHOD AND APPARATUS FOR MAKING A CLUSTER OF COMPUTERS APPEAR AS A SINGLE HOST ON A NETWORK

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to the field of clustering computers. More specifically, the invention relates to a computer cluster which appears to be a single host computer when viewed from outside the cluster, e.g. from a network of computers.

2. Description of the Prior Art

The prior art discloses many ways of increasing computing power. Two ways are improving hardware performance and building tightly coupled multiprocessor systems. Hardware technology improvements have provided an approximately 100% increase in computing power every two years. Tightly coupled systems, i.e., systems with multiple processors that all use a single real main storage and input/output configuration, increase computing power by making several processors available for computation.

However, there are limits to these two approaches. Future increases in hardware performance may not be as dramatic as in the past. Tightly-coupled multiprocessor versions of modern, pipelined and cached processors are difficult to design and implement, particularly as the number of processors in the system increases. Sometimes a new operating system has to be provided to make the tightly-coupled systems operate. In addition, overhead costs of multi-processor systems often reduce the performance of these systems as compared to that of a uniprocessor system.

An alternative way of increasing computer power uses loosely-coupled uniprocessor systems. Loosely-coupled systems typically are independent and complete systems which communicate with one another in some way. Often the loosely-coupled systems are linked together on a network, within a cluster, and/or within a cluster which is on a network. In loosely coupled systems in a cluster, at least one of the systems is connected to the network and performs communication functions between the cluster and the network.

In the prior art and also shown in FIG. 1A, clusters **100** comprise two or more computers (also called nodes or computer nodes **105** through **109**) connected together by a communication means **110** in order to exchange information. Nodes (**105** through **109**) may share common resources and cooperate in doing work. The communication means **110** connecting the computers in the cluster together can be any type of high speed communication link known in the art, including: 1. a network link like a token ring, ethernet, or fiber optic connection or 2. a computer bus like a memory or system bus. A cluster, for our purposes, also includes two or more computers connected together on a network **120**.

Often, clusters of computers **100** can be connected by various known communications links **120**, i.e., networks, to other computers or clusters. The point at which the cluster is connected to the outside network is called a boundary or cluster boundary **125**. The connection **127** at the boundary is bi-directional, i.e., there are incoming and outgoing messages at the boundary. Information which originates from a computer (also called a host or host computer) **130** that is on the network **120** outside the cluster, which then crosses the boundary **127**, and which finally enters the cluster **100** destined for one node (called a destination node) within the cluster **100**, is called an incoming message. Likewise, a message which originates from a node (called a source node) within the cluster **100** and crosses the boundary **125** destined for a host **130** on the network outside the cluster is called an outgoing message. A message from a source node within the cluster **100** to a destination also within the cluster **100** is called an internal message.

The prior art includes clusters **100** which connect to a network **120** through one of the computer nodes in the cluster. This computer, which connects the cluster to the network at the boundary **125**, is called a gatewaY **109**. In loosely-coupled systems, gateways **109** process the incoming and outgoing messages. A gateway **109** directs or routes messages to (or from) the correct node in the cluster. Internal messages do not interact with the gateway as such.

FIG. 1B shows a prior art cluster **100**, as shown in FIG. 1A, with the gateway **109** connected to a plurality (of number q) of networks **120**. In this configuration, each network **120** has a connection **127** to the gateway **109**. A cluster boundary **125** is therefore created where the gateway **109** connects to each network **120**.

FIG. 1C goes on to show another embodiment of the prior art. In this embodiment, the cluster **100** has more than one computer node (**105** through **109**) performing the function of a gateway **109**. The plurality of gateways **109**, designated as G1 through Gp each connect to one or more networks **120**. In FIG. 1C, gateway G1 connects to a number r of networks **120**, gateway G2 connects to a number q of networks **120**, and gateway Gp connects to a number s of networks **120**. Using this configuration, the prior art nodes within the cluster **100** are able to communicate with a large number of hosts **130** on a large number of different networks **120**.

All the prior art known to the inventors uses gateways **109** to enable external hosts to individually communicate with each node (**105** through **109**) in the cluster **100**. In other words, the hosts **130** external to the cluster **100** on the network **120** have to provide information about any node (**105** through **109**) within the cluster **100** before communication can begin with that node. The hosts **120** external to the cluster also have to provide information about the function running on the node which will be accessed or used during the communication. Since communication with each node (**105** through **109**) must be done individually between any external host **130** and any node within the cluster **100**, the cluster **100** appears as multiple, individual computer nodes to hosts outside the cluster. These prior art clusters do not have an image of a single computer when accessed by outside hosts. Examples of prior art which lacks this single computer image follow.

DUNIX is a restructured UNIX kernel which makes the several computer nodes within a cluster appear as a single machine to other nodes within the cluster. System calls entered by nodes inside the cluster enter an "upper kernel" which runs on each node. At this level there is an explicit call to the "switch" component, functionally a conventional Remote Procedure Call (RPC), which routes the message (on the basis of the referred to object) to the proper node. The RPC calls a program which is compiled and run. The RPC is used to set up the communication links necessary to communicate with a second node in the cluster. A "lower kernel"

running on the second node then processes the message. DUNIX is essentially a method for making computers within the cluster compatible; there is no facility for making the cluster appear as a single computer image from outside the cluster.

Amoeba is another system which provides single computer imaging of the multiple nodes within the cluster only if viewed from within the cluster. To accomplish this, Amoeba runs an entirely new base operating system which has to identify and establish communication links with every node within the cluster. Amoeba cannot provide a single computer image of the cluster to a host computer outside the cluster. Amoeba also has to provide an emulator to communicate with nodes running UNIX operating systems.

Sprite is a system which works in an explicitly distributed environment, i.e., the operating system is aware of every node in the cluster. Sprite provides mechanisms for process migration, i.e., moving a partially completed program from one node to another. To do this, Sprite has to execute RPCs each time a new node is accessed. There is no single computer image of the cluster presented to the network hosts outside these systems.

V is a distributed operating system which is able to communicate only with nodes (and other clusters) which are also running V. UNIX does not run on V.

Other techniques for managing distributed system clusters, include LOCUS, TCF, and DCE. These systems require that the operating system know of and establish communication with each individual node in a cluster before files or processes can be accessed. However, once the nodes in the cluster are communicating, processes or files can be accessed from any connected node in a transparent way. Thus, the file or process is accessed as if there were only one computer. These systems provide a single system image only for the file name space and process name space in these systems. In these systems, files and processes can not be accessed by host computers outside the cluster unless the host has established communication with a specific node within the cluster which contains the files and/or processes.

3. Statement of Problems with the Prior Art

Prior art computer clusters fail to appear as one entity to any system on the network communicating with them, i.e., the prior art does not offer the network outside its boundary a single computer image. Because of this, i.e., because computers outside the boundary of the cluster (meaning outside the boundary **125** of any gateway **109** of the cluster **100**) have to communicate individually with each computer within the cluster, communications with the cluster can be complicated. For example, computers outside the boundary of the cluster (hosts) have to know the location of and processes running on each computer within the cluster with which they are communicating. The host computers need to have the proper communication protocols and access authorization for each node within the cluster in order to establish communication. If a node within the cluster changes its location, adds or deletes a program, changes communication protocol, or changes access authorization, every host computer external to the cluster for which the change is relevant has to be informed and modified in order reestablish communication with the altered node within the cluster.

The prior art lack of a single computer image to outside host computers also limits cluster modification and reliability. If hosts try to communicate with a node within the cluster which has been removed, is being maintained, or has failed, the communication will fail. If a new node(s) is added to the cluster, i.e., the cluster is horizontally expanded, the new node will be unavailable to communicate with other host computers outside the cluster without adding the proper access codes, protocols, and other required information to the outside hosts.

Accordingly, there has been a long felt need for a cluster of computers which presents a single computer image, i.e., looks like a single computer, to computers external to the cluster (gateway) boundary. A single computer image cluster would have the capability of adding or deleting computers within the cluster; changing and/or moving processes, operating systems, and data among computers within the cluster; changing the configuration of cluster resources; redistributing tasks among the computer within the cluster; and redirecting communications from a failed cluster node to an operating node, without having to modify or notify any computer outside the cluster. Further, computers outside the cluster, would be able to access information or run processes within the cluster without changing the environment where they are operating.

Systems like DUNIX, Amoeba, Sprite, and V provide some degree of a single system image from within the cluster (i.e., within the gateway boundaries **125**) by writing new kernels (in the case of Amoeba, a totally new operating system.) This requires extensive system design effort. In addition, all the nodes of the cluster must run the system's modified kernel and communicate with servers inside the system using new software and protocols.

LOCUS, TCF and DCE provide single system images only for computers which are part of their clusters and only with respect to file name spaces and process name spaces. In other aspects, the identities of the individual nodes are visible.

OBJECTIVES

An objective of this invention is an improved method and apparatus for routing messages across the boundary of a cluster of computers to make the cluster of computers on a network appear as a single computer image to host computers on the network outside the cluster.

Also an objective of this invention is an improved method and apparatus for routing messages across the boundary of a cluster of computers to enable outside host computers on a network to use the same software and network protocols to access functions and information within the computer cluster as they would use to access those functions and information on a single remote host.

Also an objective of this invention is an improved method and apparatus for routing messages across the boundary of a cluster of computers so that computer nodes within the cluster can communicate with outside hosts on networks such that, from the viewpoint of the outside host, the communication is with a single remote host, i.e., the cluster, rather than with the individual cluster nodes.

A further objective of this invention is an improved method and apparatus for routing messages across the boundary of a cluster of computers so that work requests from outside the cluster can be evenly distributed among the computer nodes in the cluster.

SUMMARY OF THE INVENTION

This invention, called an encapsulated cluster, is a method and apparatus for routing information that crosses the boundary of a computer cluster. The information is in the form of port type messages. Both incoming and outgoing messages are routed so that the cluster appears as a single computer image to the external host. The encapsulated cluster appears as a single host to hosts on the network which are outside the cluster.

The apparatus comprises two or more computer nodes connected together by a communication link, called an interconnect, to form a cluster. (Note that in one embodiment of the invention, the interconnect can be a network.) One of the computers in the cluster, serving as a gateway, is connected to one or more external computers and/or clusters (hosts) through another communication link called a network. A gateway can be connected to more than one network and more than one node in the cluster can be a gateway. Each gateway connection to a network, i.e., boundary, has an address on the network. Each gateway has a message switch which routes incoming and outgoing messages by changing information on the message header based on running a specific routing function that is selected using port and protocol information in port type messages.

Since all incoming messages are addressed to the gateway, the cluster appears as a single computer to hosts outside the cluster that are sending incoming messages to nodes within the cluster. When processing incoming messages, the gateway first reads a protocol field in the message header and analyzes the message to determine if it is a port type message originating from a location outside the cluster. If the message is of port type, the location of the port number on the message is found. This port number and protocol type is used to search for a match to a port specific routing function in a table residing in memory within the message switch. If a table entry is matched, a routing function associated with the entry is selected and run. The routing function routes the message to the proper computer node within the cluster by altering information on the incoming message so that the message is addressed to the proper node within the cluster.

For outgoing messages, originating from a source node within the cluster, the message switch first recognizes that the message is a port type message that will cross the cluster boundary. The message switch then alters the message so that the source address is the gateway address rather than the address of the source node. In this way, computers external to the cluster perceive the message as coming from the gateway computer on the network rather than the sending node within the cluster.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGS. 1A–1C show three embodiments of prior art computer clusters that are attached to external communication links like networks.

FIG. 2 shows an embodiment of the present invention.

FIGS. 3*a*–3*e* shows the general structure of an incoming and outgoing message and a more specific message structure using the internet communication protocol.

FIG. 4 shows a preferred embodiment of a message switch.

FIG. 5 is a flow chart showing the steps performed by the present invention to route an incoming message.

FIG. 6 is a flow chart showing the steps performed by the present invention to route an outgoing message.

FIG. 7 shows data structures used by a function in the message switch which processes a MOUNT request.

FIG. 8 is a flow chart of the computer program performed by the function in the message switch which processes a MOUNT request.

FIGS. 9*a*–9*b* show data structures and a flow chart used by a function in the message switch which processes NFS requests.

FIG. 10 shows data structures used by functions in the message switch which process TCP connection service requests, in particular rlogin.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 2 shows one embodiment of an encapsulated cluster **200**, the present invention. The cluster comprises a plurality of computer nodes (**105** through **109**) one of which is a gateway **109**. The nodes are connected together by a high speed interconnect **110**, e.g., a network or any other link that is commonly used in the art. The gateway is connected with a bidirectional communication link **127** to a network **120**. A boundary **125** is defined at the connection point between the network **120** and the gateway **109**. Computers, called hosts **130**, connect to the network **120** and can communicate with nodes within the cluster by passing messages through the gateway **109**. An incoming message **210** is shown as being sent from a host **130**, passing through the cluster boundary **125**, a gateway port **230**, a gateway message switch **240**, a gateway routing function **250**, the interconnect **110**, and ultimately to the destination, the destination node **107** in the cluster **200**. In a similar manner, an outgoing message **220**, is shown originating at a source node **105** within the cluster **200**; passing through the interconnect **110**, gateway message switch **240**, gateway port **230**, cluster boundary **125**, and ultimately to the destination host **130**.

Although FIG. 2 represents a single cluster **200** with a single gateway **109**, it is readily appreciated that one skilled in the art given this disclosure could produce multiple embodiments using this invention. For example, the cluster **200** might have multiple gateways **109** each connected to one or more networks or single host computers. A single gateway **109** may also have a plurality of network connections each of which being capable of communicating with one or more external hosts or one or more external networks. All these embodiments are within the contemplation of the invention.

The encapsulated cluster **200** connects **127** to a high speed communication link **120**, here called a network **120**. Host computers **130**, also connected to the network **120**, communicate with the encapsulated cluster **200**, and the nodes (**105** through **109**) within the cluster, over the network **120**. The host computers **130** used in the invention include any general purpose computer or processor that can be connected together by the network **120** in any of the many ways known in the art. The preferred network **120** is token-ring or ethernet. However, this high speed communication link **120** might also include other connections known in the art like computer system buses. A host computer **130** could also be an encapsulated cluster of computers **200**, i.e.,

the present Invention, which gives the image of a single computer to the network 120.

Nodes (105 through 109) in the encapsulated cluster 200 can also comprise any general purpose computer or processor. An IBM RISC SYSTEM/6000 was the hardware used in the preferred embodiment and is described in the book SA23-2619, "IBM RISC SYSTEM/6000 Technology." (RISC SYSTEM/6000 is a trademark of the IBM corporation.) These nodes may be independent uniprocessors with their own memory and input/output devices. Nodes can also be conventional multiprocessors whose processors share memory and input/output resources.

Nodes (105 through 109) within the cluster are connected together by a high speed communications link called an interconnect 110. This interconnect includes any of the many known high speed methods of connecting general purpose computers or processors together. These interconnects include networks like ethernet, token rings and computer system buses like a multibus or micro Channel. (Micro Channel is a trademark of the IBM corporation.) The nodes (105 through 109) are connected to the interconnect 110 using any of the methods well known in the art. The preferred embodiment contemplated uses a fiber optic point-to-point switch as the interconnect with a bandwidth more than five times that of a token ring. A commercially available switch of this type suitable for this application is the DX Router made by Network Systems Corporation. Support software for the preferred interconnect provides a network interface between the nodes which allows the use of standard Internet Protocol (IP) communication. Internal IP addresses are assigned to the nodes of the cluster. This is a standard industry communication protocol and it allows the use of standard software communication packages.

One or more of the nodes in the cluster connects to one or more networks 120 and performs as a gateway 109. All incoming 210 and outgoing 220 messages pass through the gateway 109. The connection of the gateway 109 to the network 120 forms the boundary 125 between the encapsulated cluster 200 and the network 120. The gateway 109 also has a message switch 240 which performs operations on the incoming 210 and outgoing 220 messages. These operations enable the encapsulated cluster 200 to appear as a single computer to hosts 130 on the network 120 which are outside the cluster 200. Moreover, the gateway 109 has a plurality of routing functions 250 which operate on the incoming messages 210 in order to direct them to the correct node (105 through 109) in the cluster and to the correct communication port on that node. To reiterate, the gateway 109 may connect to one or more networks 120. An encapsulated cluster 200 may also have more than one gateway 109.

To help understand the environment of the present invention, refer now to FIG. 3 for a brief, tangential, illustrative explanation of prior art network communication protocols. Much more detail about this subject is presented in *Internetworking with TCP/IP, Principles, Protocols and Architecture*, Douglas E. Comer, Prentice Hall, which is herein incorporated by reference.

Networks of computers often comprise different kinds of communications links with different kinds of host computers connected to those links. In order for messages to be sent from one host on the link to another host on the link, rules, called protocols, are established to control the communication links, route messages, and access appropriate host computers on the link.

As shown in FIG. 3A, these protocols can be conceptually viewed as being layered, with each protocol layer making use of the services provided by the layer beneath it. The lowest layer, the Network Interface (302), deals at the hardware level, with the transmission of data between hosts on a single network of a particular type. Examples of network types are token-ring and ethernet. The Network Interface layer presents an interface to the layers above it which supports the transmission of data between two hosts on one physical network, without having to deal with the requirements of the specific network hardware being used.

The next higher layer, The Machine-to-Machine (MM) layer 304, provides the capability to communicate between hosts which are not directly connected to the same physical network. The MM layer establishes a naming system which assigns to each host computer a globally unique name (MM address). It presents an interface to the protocol layers above it which make it possible to send data to a remote host machine by simply specifying the unique MM-address of the destination host. Internet protocol (IP) is example.

The next higher protocol layer, the Port-to-Port (PP) layer 306, makes it possible for multiple processes (executing application programs) to communicate with processes at remote hosts, at the same time. The PP layer defines a set of communication ports on each host, and provides the ability to send data from a port on one machine (the source port) to a port on a remote machine (the destination port). The PP layer uses the MM protocol layer to transfer data between host machines. The PP layer presents an interface to the application layer 308 which allocates a local communication port to a process, connects that port to a remote port on a remote host, and transfers data between the local port and remote port. Examples include TCP and UDP protocols.

The application box 308 in FIG. 3A represents processes making use of the PP layer application interface in order to communicate with processes executing on remote hosts.

When an application process writes data to a communication port, the data passes down through the protocol layers on its machine, before being transferred over one or more physical networks to the destination machine. Each protocol layer prepends a protocol header to the data it is given. This is shown in FIG. 3B. The PP layer prepends a PP header 336 to the application data 337 to form a PP datagram. While different PP protocols will use headers containing different information, they will always contain the source port number and destination port number. The PP layer 306 passes to the MM layer 304 the PP datagram 330, the MM address of the destination machine, and a protocol identifier, which specifies which of the possible PP protocols is being used, (for example TCP, the value of 6, or UDP, the value of 17).

The MM layer treats the entire PP datagram 330 as data 325 and prepends to it an MM header 324 to form an MM datagram 320 or MM message 320. While the MM headers used by different MM protocols may vary, they will all contain three fields: The MM address of the sending machine (source address), the MM address of the destination machine (destination address), and the protocol identifier for the kind of PP protocol being used. The MM layer chooses an available network, and a machine on that network to send the MM packet to.

This may be the final destination machine, or an intermediate machine which will forward the MM datagram towards its final destination. The MM layer passes to the Network Interface 302 for the network to be used, the MM message, the MM address of the machine the packet is to be sent to, the identifier for the type of MM protocol that is being used.

The Network Interface 302 transmits "frames" to hosts attached to its network. It treats the entire MM message as frame data 311, and prepends to it a frame header 312 to form a frame 310. The size and format of the frame header will depend on the type of network hardware being used, e.g., token-ring frame headers will differ from ethernet frame headers. However, the frame headers must necessarily identify the most to receive the frame, and contain the identifier for the type of MM protocol being used. At this protocol level the destination host is identified by a network specific hardware address, and it is the responsibility of the Network Interface layer 302 to translate the MM address passed by the MM protocol layer to the network specific hardware address.

When a frame is accepted by the Network Interface layer 302 of the destination machine, it passes up through the protocol layers 301. Each protocol layer removes its protocol header, and passes the remaining data up to the protocol specified by the protocol identifier in its header. The PP protocol layer removes the PP header, and associates the application data with the destination port specified in the PP header. A process running on the destination machine can then access the received data by reading from that port.

FIG. 3C shows the configuration of information in a MM header 324 shown in FIG. 3B. This configuration 340 shows organization of a MM header and MM data area using Internet Protocol (IP), the MM protocol used by the preferred embodiment. The configuration 340 is shown as a sequence of 32 bit words. The first six words in the sequence are the IP header 344 (represented generally as the MM header 324 in FIG. 3B) and the remaining words 346 are the IP data area 346 (represented generally as the MM data area 325 in FIG. 3B). The numbers 342 across the top of the configuration 340 show the starting bit locations of the various fields in the words of the IP MM message. The IP fields of particular interest are the protocol 347, source IP address 348, and destination address 349 fields. Each machine using IP is assigned a globally unique IP address. The IP protocol field 347 gives information about the protocol used in the next highest layer of protocol. For instance, the protocol field 347 specifies if the next highest level will use UDP or TCP protocol. The source IP address 348 specifies the address of the computer which originated the message. The destination IP address 349 specifies the address of the computer which is to receive the message. Other fields in the IP header like total length and fragment offset are used to breakup network datagrams into packers at the source computer and reassemble them at the destination computer. The header checksum is a checksum over the fields of the header, computed and set at the source and recomputed for verification at the destination.

User Datagram Protocol (UDP) is one of two protocols used by the preferred embodiment. UDP transfers discrete packets of data, called datagrams, from a source port on one machine to a destination port on a remote machine. FIG. 3D shows the format of a UDP datagram 350 (represented generally as the PP datagram 330 in FIG. 3B). The UDP message is shown as a sequence of 32 bit words. The UDP message 350 comprises a UDP header 356, which use the first two words in the message, and a UDP data area 357, which use the remaining words in the message. The numbers 352 on top of the message 350 designate the starting location of the different fields in the UDP header 356. Since UDP is classified as a port type protocol, its header contains information about two ports, a UDP source port 353 and a UDP destination port 354. The UDP source port 353 is the port on the machine from which the message originated and the UDP destination port 354 is the port on the machine to which the message is sent.

From the viewpoint of the sending and receiving processes a UDP datagram is transferred as a unit, and can not be read by the receiving process until the entire datagram is available at the destination port. UDP uses IP to transfer a datagram to the destination machine. At the destination machine, the UDP protocol layer determines the eventual destination for the datagram using only the destination port number. This implies that within one machine, the UDP protocol layer must ensure that the UDP ports in use at any one time are unique.

Transmission Control Protocol (TCP) is the second PP protocol used in the preferred embodiment. With TCP a connection is established between TCP ports on two machines. Once the connection is established, data flows in either direction as a continuous stream of bytes. Data written on one end of a connection is accumulated by the TCP protocol layer. When appropriate, the accumulated data is sent, as a TCP datagram, to the remote machine using IP. At the remote machine, the TCP layer makes the data available to be read at the other end of the connection. Processes reading and writing data over a TCP connection do not see the boundaries between the TCP datagrams used by TCP to send data from one end of the connection to the other.

FIG. 3E shows the format of a TCP datagram. Again, the TCP format is shown as a sequence of 32 bits words, the first six words being the TCP header 366 (generically represented as 336 in FIG. 3B) and the remaining words being the TCP data 367 (generically represented as 337 in FIG. 3B). The numbers 362 across the top of the TCP format 360 represent the starting bit locations of the fields in the TCP header 366. The TCP header 366, being a port type protocol, has port information in its header 366 including a source port 363 and a destination port 364.

A TCP connection from a port on the source machine to a port on a destination machine is defined by four values: source address 348, source port number 363, destination address 340, and the destination port number 364 of the remote port on that machine. When the TCP protocol layer receives a TCP datagram, it uses all four values to determine which connection the data is for. Thus, on any one machine, TCP ensures that the set of active connections is unique. Note that this means TCP ports, unlike UDP ports, are not required to be unique. The same TCP port may be used in multiple connections, as long as those connections are unique.

Returning now to the description of the present invention, refer to FIG. 4. FIG. 4 shows a cluster of computers 200, of the present invention, routing an incoming message. The cluster 200 has three nodes (105, 106, and 109) connected together by an interconnect 110. One of the nodes is a gateway 109 which connects

to an external network 120 at a boundary 125. All messages arriving at the cluster 200, from the external network 120, arrive with the cluster gateway 109 external address as their destination IP address in their IP header. When the gateway 109 recognizes an incoming message crossing the boundary 125 it analyzes the IP header (this includes determining which protocol is designated in the protocol field). At this point, the PP datagram header is analyzed to determine the destination port of the message. The location of the destination port field will depend on the length of the IP header, and the type of PP protocol (UDP or TCP) being used. A message switch 400 in the gateway 109 uses the message protocol and the message destination port information to route the message to a node in the cluster for further processing. Note that every incoming message address only the gateway of the cluster. This gives the cluster the appearance of a single computer to the network, even though the incoming messages can be routed to any of the nodes in the cluster.

The message switch 400 comprises a message switch table 410 and the necessary software needed to route messages having a plurality of protocols and port numbers. Once the values of the destination port and protocol of the message are determined, the pair of values is looked up in the message switch table 410. (Column 412 represents values of destination ports and column 414 represents values of message protocols in the message switch table 410). For each pair of destination port and protocol values on an incoming message, there exists only one function designated f_1, f_2, . . . f_N in column 418 of the message switch table 410. This selected function, which is typically a software program, is run to determine to which node, and to which communication port on that node, the incoming message will be sent. The destination IP address is changed to the internal address of the specified node, and if necessary, the destination port is changed to the specified port number. The modified IP message is then sent to the specified node via the Network Interface for the cluster interconnect.

FIG. 5 is a flowchart description of how an incoming message is processed by the present invention.

Box 505 shows the cluster gateway waiting for a message. There are many well known ways for doing this. For example, circuitry or microcode is embodied on a device card in the gateway which connects to the network. If device card circuitry recognizes information on the frame header of packet on the network, the circuitry will store the packet in a buffer. At this point, some mechanism like an interrupt driven program in the operating system or a server will read the fields in the frame header and determine, among other things, if the packet contains a machine-to-machine (MM) protocol. If the packet contains a MM protocol, the frame header is stripped from the packet and the frame data area, i.e., the MM header and data area are processed further. This processing might include placing the MM header and data area in a queue and "waking up" a program to operate on this information.

In box 510, the destination address field in the MM header is read. Methods of reading this field are known in the art. Also, for a given MM protocol, the destination address field is a known location in the MM header. For example, in the preferred embodiment, IP is used and the destination address 349 field is positioned as the 5th 32 bit word in the IP header as shown in FIG. 3C. The destination address is designated as the value DADDR in FIG. 5.

Decision block 515 determines if the final destination of the MM information is in the cluster. This is done by comparing the destination address of the message with a list of all the cluster addresses supported by this gateway. There can be more than one cluster address when the gateway is attached to more than one network. In that case, there will be a different cluster address for each network to which the gateway is attached. The processing for the case when the comparison fails is shown in FIG. 6, and is described below. If the comparison determines that the destination address is an address in the cluster, the message is accepted for processing within the cluster. This entails an analysis of the MM header.

Box 520 begins analysis of the MM header by reading the protocol field in the MM header. In a given MM protocol, the protocol field is positioned in a pre-determined place in the header. For instance, the preferred embodiment uses IP which has its protocol field 347 starting at bit position 8 in the third 32 bit word of the header. (See FIG. 3C). The value of the protocol field in the MM header is designated as PROTO in FIG. 5.

In decision block 525 PROTO, the value of the protocol field in the MM header is compared to a list of known protocol values residing in a table or list in the gateway. If PROTO matches entries in the table which are port type protocols the process continues. If PROTO does not match entries which are port type protocols, the MM message is processed as it otherwise would be. (Note that a port type message uses protocols which require ports on both the source and destination computers in order to establish communications.)

Box 530 further analyzes the PP header by locating and reading the destination port number. This destination port, designated as DPORT in FIG. 5, is located in the PP datagram header (336 in FIG. 3B) and for TCP or UDP messages it starts at bit 16 of the first 32 bit word in the PP datagram header. (See FIGS. 3D and 3E.) However, because the MM header is still on the message, the starting position of the destination port in the datagram header may be at different locations depending upon the length of the MM header. As a result, the starting location of the destination port field has to be calculated relative to the first bit of the message. This is normally done using a header length value which is available in the MM header. In the IP case, the IP header length is field HLEN 341 which starts at the 4th bit of the 1st 32 bit word of the header. (See FIG. 3C).

In box 535, the message switch table (410 in FIG. 4) is searched for an entry which has a destination port value 412 and protocol value 414 pair equal to the respective DPORT and PROTO in the incoming message. Decision block 540 determines what action follows based on whether or not a matched pair was found.

In box 535, if the values of DPORT and PROTO do not match the port and protocol values in any entry in the message switch table, it may be possible for the message switch to compute to which node the inbound message should be sent using the value of DPORT. This is possible because of the way ports are allocated within the cluster. Processes frequently request a port to be allocated, but do not care about the particular port number of the allocated port; they let the PP protocol layer choose the number of the port which is allocated. We call such ports "non specific ports". The PP proto-

col layers on nodes of the cluster allocate non specific port numbers by an algorithm which makes it possible to compute the node address of the node a port was allocated on using only the number of the port. Given this disclosure, many such algorithms could be developed by one skilled in the art. Examples include: preallocating ranges of port numbers to the nodes, or allocating port numbers such that the number modulo the number of nodes in the cluster identifies the node.

Continuing with the flowchart of FIG. 5, if no entry is found in the message switch table which matches DPORT and PROTO (boxes 535 and 540), then decision block 545 determines if DPORT is a non specific port, i.e., one allocated by the algorithm described above. If DPORT is a non specific port, then in box 550, we compute the destination node, NODE_ADDR, from the value of DPORT. The destination port number is not changed, so NODE_PORT is set equal to DPORT.

If in box 545 it is found that DPORT is not a non specific port, then DPORT is a port unknown to the message switch. In such cases, box 545 takes the default action of having the inbound message processed by the gateway. To do this, NODE_ADDR is set to the internal node address of the gateway. The destination port is unchanged, so NODE_PORT is set to DPORT.

Decision block 555 determines how incoming messages are handled if there is a matched pair in the message switch table. The decision is based on whether or not there is a routing function (418 in FIG. 4) associated with the matched entry in the message switch table.

If there is no routing function 415 (the routing function is NULL) in the matched message switch table entry, the incoming message is processed as shown in box 560. In these cases, the new NODE_ADDR is set equal to the value in the node field (416 in FIG. 4) of the message switch table. The new NODE_PORT is again unchanged, i.e., it is set equal to DPORT. Incoming messages are processed in this way if there is only one node in the cluster which is assigned a particular port and protocol pair.

The last group of incoming messages are processed as shown in box 565. These messages have a matched pair entry in the message switch table which has a routing function designated in the table (418 in FIG. 4). These routing functions may access information which is in the MM header, PP header, and/or data fields and use this information to calculate the new destination address (NODE_ADDR) and port number (NODE_PORT). The same routing function may be used for different entries in the message switch table or the routing functions can be unique to each entry.

Message switch routing functions allow a port number and protocol pair to be used on more than one node. The need for this occurs when one wants to run an application/service which is associated with a specific well-known port number on more than one node. For example, rlogin always uses TCP port 513, and NFS always uses UDP port 2049. With the message switch, it is possible to run NFS at multiple nodes within the cluster, and have an NFS routing function, associated with UDP port 2049, to route NFS requests to the correct node. A routing function for NFS is described separately below.

Once the new NODE_PORT and NODE_ADDR values are calculated as described above, they are used to replace values in fields of the incoming message. In box 570, the destination port field (see FIG. 3D and 3E) in the PP header are changed (if necessary) to equal the value DPORT. In some protocols, other fields may have to be changed (e.g. header checksum) to maintain coherency in the header. In box 575, the destination address in the MM header is changed to equal the value of NODE_ADDR.

At this point, shown in box 580, the appropriate network protocols/headers are added to the incoming message for it to be transmitted on the interconnect. These protocols/headers are specific to the interconnect used and are well known. First, a check is made to see if the selected destination node is this gateway, by checking if NODE_ADDR is the internal address of this gateway. If it is, the inbound message is processed locally, by passing it up to the appropriate PP protocol layer in the gateway. Otherwise, the inbound message, with modified headers, is passed to the Network Interface for the cluster interconnect (110 in FIG. 1) to be sent to the selected destination node in the cluster.

FIG. 6 is a flowchart describing the processing in the gateway for MM messages which do not have a destination address equal to a valid external cluster address. We are particularly interested in the cluster processing performed for messages originating at a node in the cluster, and being sent to a machine outside the cluster. We call such messages "outbound" messages.

Box 605 shows the gateway waiting for a message. This is the identical function that is running in box 505 of FIG. 5. Note that the gateway initially does not know if the messages it receives originate from an outside host or a node within the cluster.

Box 610 is the same function as box 510 of FIG. 5. As before, the destination address (DADDR in FIG. 6) of the MM header is read. Again, for IP, the destination address resides as the 5th 32 bit word in the IP header. (See FIG. 3C).

Box 615 is the same function as box 515 in FIG. 5. It compares DADDR to a list of external cluster addresses supported by the gateway. FIG. 5 described the processing when DADDR is found in the list (the YES branch), i.e., when the message is an inbound message to the cluster. In FIG. 6 we detail the "NO" path out of box 615, for messages which arrive at the cluster gateway, but do not have a cluster address as their destination. These messages may originate from within the cluster or from outside the cluster.

In box 620 the source address (SADDR in FIG. 6) is read. For IP, the source address is found in the IP header as the 4th 32 bit word. (See FIG. 3C).

Box 625 determines if the message is an outgoing message. An outgoing message must have originated at a node within the cluster (SADDR is the address of a cluster node) and be destined for a host outside the cluster (DADDR is the address of a host outside the cluster). If either of these conditions is not satisfied, i.e., the message is not an outgoing message, the message is processed at the frame level in box 640.

However, if the message is an outgoing message type, it is processed in box 630 before going on the network. In box 630, the source address in the message header (SADDR) is changed to that of the address of the cluster. The cluster address for this purpose is the (or an) address of the gateway where the message will be placed on the network. By changing the source address in this way, hosts on the network external to the cluster will view the message as coming from the gateway and not the source node within the cluster. As a result, the source node will be invisible to the external host and the

entire cluster will have the image of a single computer, whose address is the gateway connection address. At this point, the outgoing message is ready for frame level processing in box 640. Note that if the source port number of an incoming message was changed by the message switch when it entered the cluster, the message switch must change this source port number to one of the port numbers on the gateway when the message leaves the cluster. This insures that the cluster appears as a single image computer to hosts on the network.

Box 640 performs the frame level processing. This is done using any number of known methods in the art. Specifically, a standard subroutine is called which incorporates the MM message into a frame data area, adds a frame header to the frame data area, and places the newly created frame message on the network. The subroutine is designed specifically to support the type of network upon which the message is placed. For example, messages which return to nodes within the cluster (DADDR is a cluster node address) will be processed by a subroutine that supports the protocol used by the interconnect 110. Alternatively, messages which are placed on an outside network will be processed by a subroutine that supports that particular outside network.

An alternative preferred embodiment is now described. In this embodiment, all nodes have a list of all of the cluster addresses supported by the gateways. Any node in the cluster receiving a message compares the destination address of the message to the list of addresses. If the message is addressed to any cluster address in the list, or the nodes own internal address, the node receiving the message will process the message.

In this embodiment, the cluster gateways process inbound MM messages as described by the flowchart in FIG. 5, except that the destination address in the MM header is not replaced by NODE_ADDR (the internal address of the node to which the message is to be forwarded). Processing is as in FIG. 5, but with box 575 deleted. NODE_ADDR is still computed, and still used (box 580 of FIG. 5) to specify the node to which an inbound MM message is to be sent. Frame headers used by the Network Interface layer for the cluster interconnect still use the internal address of the destination node. In this alternate embodiment, forwarded MM messages arrive at the selected node with the destination address field in the MM header containing the external cluster address it had when it arrived at, and was accepted by, the cluster gateway. (When using IP as the MM protocol, this means that IP incoming messages arrive at the nodes with the destination IP address being the external IP address of the cluster, rather than the internal IP address of the receiving node.)

In this alternate preferred embodiment, the nodes perform additional cluster processing for outbound MM messages. A node detects when it is originating (is the source of) a MM message to a destination outside the cluster. Instead of using its own internal address, the node chooses an external cluster address, and places it in the source address field of the MM header. If the cluster has only one address, because it is attached to only one network at one gateway, then the node uses that address as the source address, and sends the message to that gateway for forwarding to the external destination. If the cluster is attached to multiple networks via multiple gateways, the cluster will have a different cluster address for each of these networks. Then, the nodes choose the cluster address associated with the network that provides the most direct path to the external host, and the nodes send the outbound MM message to the gateway attached to the selected network. One network is designated as a default, to be used when a node is not able to choose among the multiple attached networks.

The cluster processing done in the nodes for outbound messages replaces the cluster processing done for outbound messages in the gateways in the first preferred embodiment, previously described in FIG. 6. In this alternate embodiment, boxes 620, 625, and 630 in FIG. 6 are not necessary. In this embodiment, gateways receive outbound messages from nodes which have a cluster address in the MM header source address field (rather than the address of the node originating the node). This is not a problem, since only the destination address is needed by the gateway to correctly forward the outbound message via one of the attached networks.

This alternative embodiment allows an external host to communicate with a specific node in the cluster if the node address is known. The message switch mechanism is by-passed because the NO path from box 515 of FIG. 5 will be taken. This embodiment can be used by a utility management function running outside the cluster that could get usage statistics from a particular node or start or stop application processes on a particular node.

To further illustrate the function of the present invention the following non limiting examples are presented.

NFS and MOUNT Servers: Overview

Network File System (NFS) is a software package developed by Sun Microsystems which allows host machines to access filesystems or files from other machines on the network. The protocol begins with a MOUNT request to the machine (called the SERVER) containing the desired filesystem. The requesting machine (the CLIENT) receives a filehandle (FH) as the result of a successful MOUNT request. The CLIENT delivers a FH as part of subsequent NFS requests, and may receive other FH's and/or data as a result of successful NFS requests.

A CLIENT can MOUNT from a plurality of SERVERS at any given time. Using the current invention, a physical plurality of servers, i.e., nodes of the cluster, can be used by CLIENTS outside the cluster as if the plurality were a single SERVER. Alternately, the plurality of nodes of the cluster can be considered a single distributed NFS SERVER, thereby obtaining the benefits of load distribution, and also continued provision of service in the event of a failure of one server, if the version of NFS which provides that capability (i.e., HA/NFS, an IBM product) is executed in the cluster.

EXAMPLE 1

MOUNT server

MOUNT is a part of the Network File System (NFS) suite of services. When it starts up, it obtains a privileged port and registers itself, i.e., its program number and version, and port number, with PORTMAPPER, a service whose function it is to provide to external clients the port on which a service, in this example MOUNT, receives messages.

At each node of the cluster on which MOUNT is running, the above sequence occurs. In general, MOUNT's port at each node of the cluster will be different from MOUNT's port at other cluster nodes, including the gateway. Clients of MOUNT outside the cluster will receive from PORTMAPPER at the gate-

way MOUNT's port in the gateway. But, as part of this embodiment, MOUNT's port at each cluster node on which it is running is communicated to and remembered by the message switch in the gateway.

Once the client running at the host receives the MOUNT port number of the node, the client at the host sends another (incoming) message to the cluster which accesses the MOUNT function. This is treated as an incoming message as described below. When a MOUNT request succeeds, the client on the host receives from the server a filehandle. The filehandle is a 32 byte token, opaque to the client, which is used for subsequent access to the mounted directory.

FIG. 7 shows a MOUNT MM message and two data structures which exist as part of the MOUNT function support in the cluster, i.e., a function which would appear in the message switch table 410 column 418 in FIG. 4.

FIG. 7 shows the structure of an incoming MOUNT request 700. Since MOUNT requests are IP/UDP type messages, the MM header 324 (see also FIG. 3) is an IP header and the datagram header 336 is a UDP header. In the datagram data area 337, the MOUNT request contains a character string which represents the filename (or filesystem name) 715 which is to be MOUNTED. This filename 715 is read by the MOUNT function (f_MOUNT) in the message switch table 410 and is used to access node information 725 from a cluster export table 720 present in the f_MOUNT function. The node information 725 in the Cluster Export Table 720 gives the node number of the node on which the file identified by the filename 715 resides. This filename 715 (the filename in FIG. 7 is /Projects) can represent an individual file or a directory with a number of files. The node information (number) 725 is then used to access a Mount Port Table 730 in the f_MOUNT function. The Mount Port Table 730 uniquely matches the node number of a node to the port number 735 being used by the MOUNT program running on that node.

FIG. 8 is a flowchart showing how the f_MOUNT function in the message switch processes an incoming MOUNT request message. This function is performed in general terms in box 565 of FIG. 5. Note that an outgoing MOUNT request is not covered because it is handled the same way any other outgoing message is handled as previously described. See FIG. 6.

In box 805, the f_MOUNT function locates and reads the filesystem name (FSN) 715 in the incoming MOUNT request 700. Next, in box 810, the Cluster Export Table 720 is searched for an entry 722 which matches the FSN or contains the FSN. Decision block 815 determines if a match has been found between the FSN and an entry 722 in the Cluster Export Table 720. If no entry matches, the f_MOUNT function returns a "ret_code" of not OK 820. This ret_code indicates that there is an error, i.e., that the requested FSN is not available for mounting from the cluster. In these cases, the gateway tries to process the MOUNT request. If a match is found in the Cluster Export Table 720, the corresponding node number, N, 725 is read (box 825). This node number is used to search the Mount Port Table 730 (box 830) for an entry matching the node number. That mount port number, P, 735 is read from the table. In box 840, the f_MOUNT function returns a NODE_ADDR variable equal to the matched node number, N, and a NODE_PORT variable equal to the mount port number, P. A ret_code of OK is also returned (box 850).

The message switch forwards the mount request to node N, where it is processed by the mount server on that node. If successful, the mount server creates and returns to the requester a filehandle (FH) for the file specified in the mount request. As part of the support for the NFS message switch routing function, all filehandles generated within the cluster, contain in a previously unused field, the node number, N, of the node on which the associated file resides.

EXAMPLE 2

NFS server

An NFS request is an IP/UDP type request which provides a filehandle (obtained using a MOUNT request or from a previous NFS request) to access a file represented or in a directory represented by the filehandle.

FIG. 9 shows the structure of a NFS request 900 and a flow chart showing how the f_NFS, the NFS function, processes a NFS request.

As with any IP/UDP request, the MM datagram has an IP header 324 and an UDP header 336. The NFS datagram data area 337 contains a filehandle 915 which has many fields. One of these fields, normally unused, in the present invention contains the cluster node address, N, of the file being accessed. Other fields 925 contain NFS file handle data.

The NFS function, f_NFS, first locates and reads the NFS file handle (FH) 915 in the NFS request (box 930). Next, in box 935, the NFS function locates and reads the node number, N, in the NFS filehandle, which was inserted by the preferred embodiment MOUNT request (described above). In box 940, the return variable NODE_ADDR is set equal to N and the variable NODE_PORT is set equal to 2049. The value of 2049 is a well known number for NFS requests and is the same value used for all nodes in the cluster. For incoming NFS requests, the process shown in FIG. 9 is performed as part of the process shown in box 565 in FIG. 5. Outgoing NFS requests are handled as all outgoing messages as explained above. See FIG. 6.

EXAMPLE 3

TCP-Based Servers

RLOGIN, REXEC, RSH, and TELNET are examples of TCP connection based services associated with well known port numbers which are listed in a known IP file called /etc/services. Using these protocols, clients running on external hosts establish connections with the well known port associated with the service that they want. Except for the value of the well known port number, the treatment of these and other similar protocols is the same so this discussion is limited to describing RLOGIN whose well known port number is 513. FIG. 10 illustrates the data structures and control flow for RLOGIN for a cluster with a single gateway.

On each node of the cluster, including the gateway, the normal rlogin daemon is started. A daemon in the UNIX context is a program which provides essential services for users of its function, and which is normally executing and available to its users whenever its host system is available. Each daemon "listens" on port 513, waiting to "accept" connection requests from rlogin clients running on hosts outside the cluster. While there are multiple occurrences of port 513 and the associated rlogin daemon within the cluster, when viewed from the internet, the cluster appears as a single host with a

single rlogin daemon. This is accomplished because the only port 513 seen outside the cluster is that of the cluster gateway.

A request for rlogin to the cluster is addressed to the gateway and arrives at the gateway with destination port number 513 and protocol field value TCP, i.e., 6. In the message switch, port number 513 and protocol TCP are matched with function f_inconn 1006, which is invoked. A flag indicator in the message specifies that this is a request for a new connection. The f_inconn function finds the source address (s_addr) 1022 and the source port number (s_port) 1024 in the incoming message, and compares this pair of values against entries in a Cluster Connection Table 1020. If it finds no matching entry (the normal case), it creates one, and associates with it a node 1026 according to a load distribution, i.e., balancing, algorithm. (One preferred algorithm is round-robin but any other load balancing algorithm know in the art can be used.) The message is forwarded to the chosen node where the connection is established by the rlogin daemon running on that node. If there is an existing matching entry in the Cluster Connection Table 1020, the message is forwarded to the associated node. This is likely an error, and the rlogin daemon on the node will generate the appropriate error response.

Subsequent messages associated with an established connection are also processed by the connection manager, f_inconn. s_addr and s_port are used to find the matching entry in the Cluster Connection Table for the connection, and the messages are forwarded to the node associated with the connection.

If a message does not contain a flag indicator for a request to establish a connection and a matching entry in the Cluster Connection Table is not found, the message is discarded.

An incoming message may contain flag indicators which specify that the sending host machine intends to terminate the connection. When this occurs, the entry for the connection in the Cluster Connection Table is removed, and the message is then forwarded to the node associated with the connection, which performs protocol-level processing associated with terminating the connection.

It will be appreciated that the examples given above of this invention are for illustrative purposes and do not limit the scope of the invention. One skilled in the art given this disclosure could design an encapsulated cluster which would appear as a single image computer to outside hosts on a network for a variety of types of port type incoming and outgoing messages. For example:

As shown above, the encapsulated cluster can be accessed by other hosts on a network as if it were a single machine on the network. Hosts not part of the cluster operate in their normal network environment without modification while accessing services and data from the encapsulated cluster.

As shown above, services associated with a protocol port (NFS, MOUNT, RLOGIN) can be distributed over multiple nodes of the cluster, and the message switch can be used to distribute the workload among the multiple instances of the service.

Furthermore, the message switch facilitates the use of the cluster for high availability applications. One can run two instances of an application, a primary and a backup, on different nodes of the cluster, where the backup is capable of taking over for the primary if it fails. If failure of the primary occurs, the message switch can be used to direct messages that would have gone to the primary, instead to the node running the backup, by changing the message switch table entries for the ports used by the application. Hosts outside the cluster continue to communicate with the application, using the same address and ports as if the failure did not occur.

Similarly, the single system image provided by the encapsulated cluster via the message switch supports the capability for applications which are able to communicate between separate instances of themselves to transparently take over the workload of a failed instance, if these multiple instances are executing in encapsulated cluster nodes. If surviving instances are able to continue work that had been executing in a failed node, the message switch can be configured to direct messages which would have been sent to the failed node, instead to the node taking over the failed node's workload. To hosts outside the cluster, the application continues as if the failure did not occur.

We claim:

1. A method for routing incoming messages across a boundary of a cluster of computer nodes, the cluster connected to one or more networks, comprising the steps of:

reading a software communication protocol number in a message header of the message to recognize an incoming message as a software communication protocol port type message, the message having a destination address of a gateway node within the cluster of computer nodes;

locating and reading a software communication protocol port number in the message header of the software communication protocol port type message;

matching both the software communication protocol port number and the software communication protocol number to an entry in a message switch memory, the matched software communication protocol port number entry being associated with a software communication protocol port specific function which selects a routing destination for the message from a plurality of possible destinations, the destination being a computer node in the cluster; and

routing the message to the computer node destination.

2. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 1, where the function modifies the destination address in the message header to that of the address of the destination node selected by the function.

3. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 2, where the function also modifies a destination port number in the message header to that of a port number on the destination node.

4. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 1, where there is no matched entry in the message switch memory and the destination node is computed by a routing algorithm.

5. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 4, where the algorithm determines the destination node address by using the destination port number in the message header.

6. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 1, where

messages are sent to the selected destination node without changing the destination address field in the message header, and the destination node accepts any message having a destination address recognized by a cluster gateway message switch.

7. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 1, where the message which is to be routed to a failed node is routed to an operating node in the cluster.

8. A method for routing incoming messages across a boundary of a cluster of computer nodes, the cluster connected to one or more networks, comprising the steps of:

reading a software communication protocol number in an IP message header to recognize the incoming message as a software communication protocol port type message, the IP message header being on a message with a destination address of a gateway within the cluster of computer nodes;

locating and reading a software communication protocol port number in the message header of the software communication protocol port type message;

matching both the software communication protocol port number and the software communication protocol number to an entry in a message switch memory, the matched software communication protocol port number entry being associated with a port specific function which selects a routing destination for the message from a plurality of possible destinations, the destination being a computer node in the cluster; and

routing the message to the computer node destination.

9. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 8, where the function modifies the destination address in the message header to that of the address of the destination node selected by the function.

10. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 9, where the function also modifies the destination port number in the IP message header to that of a port number on the destination node.

11. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 8, where there is no matched entry in the message switch memory, and the destination is computed by a routing algorithm.

12. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 11, where the algorithm determines the destination node address by using the destination port number in the message header.

13. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 8, where the protocol number is a UDP identifier.

14. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 13, where the UDP protocol number is 17.

15. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 8, where there is more than one destination node in the cluster using a common port number.

16. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 15, where the common port number is the port number associated with NFS.

17. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 15, where the common port number is the port number associated with RLOGIN.

18. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 8, where the message is a MOUNT type request for a filehandle and part of the message is a data field which contains a filesystem name that was requested by the source host of the MOUNT request.

19. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 18, where the selected node returns the filehandle which includes a destination node identifier used by a later NFS request.

20. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 8, where the message is a NFS type request and the message data field contains a filehandle.

21. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 20, where the filehandle includes a destination node identifier.

22. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 8, where the protocol number is a TCP identifier.

23. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 22, where the TCP protocol number is 6.

24. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 22, where the function is a connection manager which routes the message to a destination computer node in the cluster based on the values of a source port number and a source IP address number.

25. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 24, where the destination computer node for new connections is determined by algorithm.

26. A method of routing incoming messages across a boundary of a cluster of computers, as in claim 25, where the algorithm determines the destination computer node in a way to balance a load among nodes in the cluster.

27. A method of routing incoming messages across a boundary of a cluster of computer, as in claim 26, where the algorithm is a round-robin.

28. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 25, where the algorithm ensures that, for each external source host, all TCP messages from that source host are routed to the same destination node.

29. A method of routing incoming messages across the boundary of a cluster of computers, as in claim 8, where messages are sent to the selected destination node without changing the destination IP address field in the message header, and the destination node accepts any message having a destination IP address recognized by a cluster gateway message switch.

30. A method of routing incoming messages, as in claim 12, where the software communication protocol number in the IP message header is a TCP protocol number and the software communication protocol port number is a TCP port number.

31. A method of routing an incoming TCP connection based message across a boundary of a cluster of computers, as in claim 30, where the message is a RLOGIN type.

32. A method of routing an incoming TCP connection based message across a boundary of a cluster of computers, as in claim 30, where the message is a REEXEC type.

33. A method of routing an incoming TCP connection based message across a boundary of a cluster of computers, as in claim 30, where the message is a RSH type.

34. A method of routing an incoming TCP connection based message across a boundary of a cluster of computers, as in claim 30, where the message is a TELNET type.

35. An apparatus for routing messages across a boundary of a cluster of computers on a network, comprising:

two or more computers connected together by an interconnect in a cluster, each computer being a node of the cluster;

one or more computer nodes in the cluster being a gateway node to the cluster, the gateway node being connected to the network at a cluster boundary and having a software communication protocol address on the network, the protocol address being the address of the gateway and the cluster;

a message switch, residing in the gateway node which matches both a message software communication protocol number and destination software communication protocol port number of the message coming into the cluster from the network to an entry in a table in the message switch memory; and

one or more routing functions, residing in the message switch table, one of the routing functions associated with the matched table entry which is used for directing messages crossing the cluster boundary into the cluster to a destination node within the cluster, the destination node being determined by the message switch using information on the message.

* * * * *